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

Control-flow Traversal
Translation Rules
AEMgr
AbstractExecution
Automated Translation

Lab-Exercise-3
Manual Translation

Assignment-3

Assertion Verification
Buff-overflow Detection

Safe
Unsafe

*.*.ll

Software Security Analysis 2024
https://github.com/SVF-tools/Software-Security-Analysis
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 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.

Concrete

Abstract

...  -3  -2  -1

0

1  2  3  ...

\{ - \}

\{ 0 \}

\{ + \}
Abstract Interpretation: Levels of Abstractions

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

\[ x = 0 \text{ or } 2 \]

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.

\[ x = 0 \text{ or } 2 \]

Concrete: \( \ldots -3 \ -2 \ -1 \)

Abstract: \( \{0, 2\} \)

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.

What is the abstract value?

Concrete

\[ x = 0 \text{ or } 2 \]

Abstract

\[ \ldots -3 \quad -2 \quad -1 \]

\[ 0 \quad 1 \quad 2 \quad 3 \quad \ldots \]

\[ [0, 2] \quad \{0, 2\} \]

Software Security Analysis 2024

https://github.com/SVF-tools/Software-Security-Analysis
Abstract Interpretation: Levels of Abstractions

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

Concrete

\[ x = 0 \text{ or } 2 \]

Abstract

\[ \ldots, -3, -2, -1 \]

What is the abstract value?

1 2 3 \ldots

\[ \{0, +\} \]

\[ [0, 2] \]

\[ \{0, 2\} \]
Abstract Interpretation: Levels of Abstractions

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

Concrete

Abstract

\[ x = 0 \text{ or } 2 \]

What is the abstract value?

\[ \ldots -3 -2 -1 \]

Precise

Software Security Analysis 2024

https://github.com/SVF-tools/Software-Security-Analysis
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.

• 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.

https://github.com/SVF-tools/Software-Security-Analysis
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)

\[
\begin{align*}
\ell_1 & \text{ void analyzeThis(int } x) \{ \\
\ell_2 & \quad \text{ int sum = 0; } \\
\ell_3 & \quad \text{ for (int } i = 0; i < x; \text{ ++i) } \{ \\
\ell_4 & \quad \quad \text{ sum += i; } \\
\ell_5 & \quad \} \\
\ell_6 & \}
\end{align*}
\]

Abstract Interpretation vs. Symbolic Execution

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

```c
void analyzeThis(int x) {
    int sum = 0;
    for (int i = 0; i < x; i++) {
        sum += i;
    }
    sum = ?
}
```

Abstract Interpretation

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

Software Security Analysis 2024

https://github.com/SVF-tools/Software-Security-Analysis
Abstract Interpretation vs. Symbolic Execution

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

\[ \text{void analyzeThis(int } x \text{)} \{
  \text{int sum = 0;}
  \text{for (int } i = 0; i < x; ++i) \{
    \text{sum += } i;
  \}
  \text{sum = ?}
\]

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

Path Answer
0

<table>
<thead>
<tr>
<th>Path</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_6 )</td>
<td>0</td>
</tr>
</tbody>
</table>

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
Abstract Interpretation vs. Symbolic Execution

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

Abstract Interpretation

\[
\begin{align*}
\ell_1: & \quad \text{void analyzeThis(int } x) \{ \\
\ell_2: & \quad \text{int } sum = 0; \\
\ell_3: & \quad \text{for (int } i = 0; i < x; ++i) \{ \\
\ell_4: & \quad \text{sum += } i; \\
\ell_5: & \quad \} \\
\ell_6: & \quad \text{sum = ?}
\end{align*}
\]

\{0, +\}

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

Symbolic Execution

\[\ldots 2 4 5 \ldots \]
\[\ldots 0 1 3 6 \ldots n \ldots \]
\[\ldots -5 -4 -3 -2 -1 \]

Path Answer

<table>
<thead>
<tr>
<th>Path</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_1 \rightarrow \ell_2 \rightarrow \ell_3 \rightarrow \ell_6$</td>
<td>0</td>
</tr>
<tr>
<td>$\ell_1 \rightarrow \ell_2 \rightarrow \ell_3 \rightarrow \ell_4 \rightarrow \ell_5 \rightarrow \ell_6$</td>
<td>1</td>
</tr>
</tbody>
</table>
Abstract Interpretation vs. Symbolic Execution

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

void analyzeThis(int x) {
  int sum = 0;
  for (int i = 0; i < x; ++i) {
    sum += i;
  }
  sum = ?
}

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

<table>
<thead>
<tr>
<th>Path</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_6$</td>
<td>0</td>
</tr>
<tr>
<td>$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4 \rightarrow l_5 \rightarrow l_6$</td>
<td>1</td>
</tr>
<tr>
<td>$l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4 \rightarrow l_5$</td>
<td>3</td>
</tr>
</tbody>
</table>

Software Security Analysis 2024

https://github.com/SVF-tools/Software-Security-Analysis
Abstract Interpretation vs. Symbolic Execution

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

Abstract Interpretation

\[
\text{void analyzeThis}(\text{int } x) \{
\text{int sum} = 0;
\text{for (int } i = 0; i < x; ++i) \{
\text{sum} += i;
\text{}}
\text{sum} = ?
\}\]

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

Symbolic Execution

<table>
<thead>
<tr>
<th>Path</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>(l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_6)</td>
<td>0</td>
</tr>
<tr>
<td>(l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4 \rightarrow l_5 \rightarrow l_6)</td>
<td>1</td>
</tr>
<tr>
<td>(l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow l_4 \rightarrow l_5 \rightarrow l_3 \rightarrow l_4 \rightarrow l_5 \rightarrow l_6)</td>
<td>3</td>
</tr>
<tr>
<td>......</td>
<td>infinite paths!</td>
</tr>
</tbody>
</table>

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
Abstract Interpretation vs. Symbolic Execution

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

```
void analyzeThis(int x) {
  int sum = 0;
  for (int i = 0; i < x; ++i) {
    sum += i;
  }
  sum = ?
}
```

**Abstract Interpretation**

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
0
1
3
......

...... infinite paths!

Path | Answer
--- | ---
$L_1 \rightarrow L_2 \rightarrow L_3 \rightarrow L_6$ | 0
$L_1 \rightarrow L_2 \rightarrow L_3 \rightarrow L_4 \rightarrow L_5 \rightarrow L_6$ | 1
$L_1 \rightarrow L_2 \rightarrow L_3 \rightarrow L_4 \rightarrow L_5$ | 3

Software Security Analysis 2024
https://github.com/SVF-tools/Software-Security-Analysis
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

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

x = ?

Approximation!

<table>
<thead>
<tr>
<th>x</th>
<th>{+, [1, 3]}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1, 3]</td>
</tr>
<tr>
<td>3</td>
<td>{1, 3}</td>
</tr>
</tbody>
</table>

Software Security Analysis 2024
https://github.com/SVF-tools/Software-Security-Analysis
Abstract Interpretation: A Code Example

```c
if (cond) {
    x = 1;
} else {
    x = 3;
}
```

Approximation!

<table>
<thead>
<tr>
<th>x</th>
<th>{+}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[1, 3]</td>
</tr>
<tr>
<td></td>
<td>{1, 3}</td>
</tr>
</tbody>
</table>

Coarse-grained but faster
Abstract Interpretation: A Code Example

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

x = ?

Approximation!

x \{+\}

x \[1, 3\]

x \{1, 3\}

Fine-grained but slower
Concrete Domain and Abstract Domain: Formal Definition

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

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

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: $a \sqsubseteq b \iff a \subseteq b$. E.g., $\{+\} \subseteq \{0, +\} \iff \{+\} \subseteq \{0, +\}$.

• 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 $\{x \in \mathbb{Z} \mid x > 0\}$, a superset 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 \( \alpha : \mathbb{C} \to \mathbb{A} \) maps a set of concrete values to its abstract ones;
- Concretization function \( \gamma : \mathbb{A} \to \mathbb{C} \) maps a set of abstract values to concrete ones.

**Example: Abstraction/concretization functions on sign domain**

<table>
<thead>
<tr>
<th>( \gamma \text{Sign}(\top) = \mathbb{Z} )</th>
<th>( \alpha \text{Sign}(c) = {+} ) if ( c \in \mathbb{Z}_{&gt;0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gamma \text{Sign}({\top}) = {x \mid x &lt; 0} )</td>
<td>( \alpha \text{Sign}(c) = {-} ) if ( c \in \mathbb{Z}_{&lt;0} )</td>
</tr>
<tr>
<td>( \gamma \text{Sign}({\top}) = {x \mid x &gt; 0} )</td>
<td>( \alpha \text{Sign}(c) = {+, 0} ) if ( c \in \mathbb{Z}_{\geq 0} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Galois Connection of Sign Domain

(a) Powerset integer domain

\[ \gamma_{\text{Sign}}(\top) = \mathbb{Z} \]
\[ \gamma_{\text{Sign}}(\{-,0\}) = \{x \mid x \leq 0\} \]
\[ \gamma_{\text{Sign}}(\{+,0\}) = \{x \mid x \geq 0\} \]

\[ \gamma_{\text{Sign}}(\emptyset) = \emptyset \]

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

(b) Sign domain

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
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
  \[ \mathbb{L}_{interval} = \langle \mathbb{I}, \subseteq, \cap, \cup, \bot, \top \rangle, \text{ where } \mathbb{I} = \{[a, b] \mid a, b \in \mathbb{Z} \cup \{-\infty, +\infty\}\} \cup \{\bot\}. \]
- Partial order: \([a_1, b_1] \subseteq [a_2, b_2] \iff a_2 \leq a_1 \land b_1 \leq b_2.\]
  - E.g., \([0, 0], [0, 1] \in \mathbb{A}_{interval}, \text{ satisfying } [0, 0] \subseteq [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
  \[ \mathbb{L}_{\text{interval}} = \langle \mathbb{I}, \subseteq, \cap, \cup, \bot, \top \rangle, \text{ where } \mathbb{I} = \{[a, b] \mid a, b \in \mathbb{Z} \cup \{-\infty, +\infty\}\} \cup \{\bot\}. \]

- Partial order: \([a_1, b_1] \subseteq [a_2, b_2] \iff a_2 \leq a_1 \land b_1 \leq b_2.\]
- E.g., \([0, 0], [0, 1] \in \mathbb{A}_{\text{interval}},\) satisfying \([0, 0] \subseteq [0, 1].\)

Given \(a_1 = [3, 8]\) and \(a_2 = [7, 12].\)

**Meet operation** \(a_1 \cap a_2\) returns the greatest Lower Bound (GLB):
- GLB = \([7, 8]\), the largest range that is shared by both \(a_1\) and \(a_2\).

**Join operation** \(a_1 \cup a_2\) 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 \(\bot\) respectively.
Figure: Powerset integer domain $\mathcal{C}$ and its abstraction as the interval domain $\mathbb{A}_{\text{interval}}$. 

Galois Connection between $\mathcal{C}$ and $\mathbb{A}_{\text{interval}}$
Abstract State and Abstract Trace

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

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

- **An abstract trace** $\sigma \in \mathbb{L} \times \mathcal{V} \to \mathbb{A}$ represents a list of abstract states before ($\ell$) and after ($\ell$) each program statement $\ell$ (\texttt{preAbsTrace} and \texttt{postAbsTrace} in Assignment-3).

<table>
<thead>
<tr>
<th>Notation</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract trace</strong></td>
<td>$\sigma$</td>
</tr>
<tr>
<td><strong>Abstract state at program point</strong> $L \in \mathbb{L}$</td>
<td>$\sigma_L$ $\mathcal{V} \to \mathbb{A}_{Interval}$</td>
</tr>
<tr>
<td><strong>Abstract value of $x$ at program point</strong> $L \in \mathbb{L}$</td>
<td>$\sigma_L(x)$ $\mathbb{A}_{Interval}$</td>
</tr>
</tbody>
</table>
Abstract Trace: A Simple Example

Control Flow Graph

\[ \ell_1 : a = 0; \]

Program point immediately after program statement \( \ell_1 \)

\[ \ell_2 : b = 1; \]

\[ \ell_3 : \ldots \]

Abstract Trace \( \sigma \)

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
Abstract Trace: A Simple Example

\[ \ell_1 : a = 0; \]

Program point immediately after program statement \( \ell_1 \)

\[ \ell_2 : b = 1; \]

Program point immediately before program statement \( \ell_2 \)

\[ \ell_3 : \ldots \]

Control Flow Graph

Abstract Trace \( \sigma \)

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
Abstract Trace: A Simple Example

\[ \ell_1 : a = 0; \quad \ell_2 : b = 1; \quad \ell_3 : \ldots \]

Program point immediately after program statement \( \ell_1 \)

Program point immediately before program statement \( \ell_2 \)

Control Flow Graph

Abstract Trace \( \sigma \)

https://github.com/SVF-tools/Software-Security-Analysis
Abstract Trace: A Simple Example

\[ \ell_1: a = 0; \]
\[ \ell_2: b = 1; \]
\[ \ell_3: \ldots \]

Control Flow Graph

Abstract state \( \sigma_{\ell_1} \):

<table>
<thead>
<tr>
<th></th>
<th>( [0, 0] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>( \bot )</td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

Abstract Trace \( \sigma \)
Abstract Trace: A Simple Example

Control Flow Graph

Abstract Trace $\sigma$

Abstract state $\sigma_{l_1}$

$$\begin{array}{c|c}
    a & [0, 0] \\
    b & \perp
  \end{array}$$

Abstract state $\sigma_{l_2}$

$$\begin{array}{c|c}
    a & [0, 0] \\
    b & [1, 1]
  \end{array}$$

$l_1: a = 0;$

$l_2: b = 1;$

$l_3: \ldots$
Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>(\sigma_{l_1}(a))</th>
<th>(\sigma_{l_2}(a))</th>
<th>(\sigma_{l_3}(a))</th>
<th>(\sigma_{l_4}(a))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(l_1) : (a = 0;)</td>
<td>(l_1)</td>
<td>(l_2) : (a &lt; 10)</td>
<td>(a &lt; 10)</td>
<td>(l_3) : (a++;)</td>
</tr>
<tr>
<td>(l_4) : (\ldots)</td>
<td>(l_4)</td>
<td>(l_2)</td>
<td>(l_3)</td>
<td>(l_4)</td>
</tr>
</tbody>
</table>

Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

Control Flow Graph

What is the abstract state after analyzing each statement?
Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$\ell_1 : a = 0;$</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \geq 10$</td>
<td>$\ell_1$</td>
<td>$\ell_2 : a &lt; 10$</td>
<td></td>
</tr>
<tr>
<td>$a &lt; 10$</td>
<td>$\ell_2$</td>
<td>$\ell_3 : a + +;$</td>
<td></td>
</tr>
<tr>
<td>$\ell_4$</td>
<td></td>
<td>$\ell_3$</td>
<td></td>
</tr>
</tbody>
</table>

What is the abstract state after analyzing each statement?

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

Control Flow Graph

$F_1, \ldots, F_4$ are transfer functions which indicate how abstract states are updated.
Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>( \sigma_{\ell_1}(a) )</th>
<th>( \sigma_{\ell_2}(a) )</th>
<th>( \sigma_{\ell_3}(a) )</th>
<th>( \sigma_{\ell_4}(a) )</th>
</tr>
</thead>
</table>

What is the abstract state after analyzing each statement?

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

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

\( F_1, \ldots, F_4 \) are transfer functions which indicate how abstract states are updated.

Control Flow Graph

\[ \ell_1 : a = 0; \]

\[ a \geq 10 \quad \ell_1 \]

\[ a < 10 \quad \ell_2 \]

\[ \ell_2 : a < 10 \]

\[ \ell_3 : a++; \]

\[ \ell_4 : \ldots \]

\[ \ell_4 \]

\[ \ell_3 \]
### Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### What is the abstract state after analyzing each statement?

- $l_1 : a = 0$; $\sigma_{\ell_1}(a) := F_1() = [0, 0]$
- $l_2 : a < 10$; $\sigma_{\ell_2}(a) := F_2(\sigma_{\ell_1}, \sigma_{\ell_3}) = \sigma_{\ell_1}(a) \sqcup \sigma_{\ell_3}(a)$
- $l_3 : a++$; $\sigma_{\ell_3}(a) := F_3(\sigma_{\ell_2}) = ([-\infty, 9] \cap \sigma_{\ell_2}(a)) + [1, 1]$

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

![Control Flow Graph](image-url)
Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What is the abstract state after analyzing each statement?

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

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

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

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

$F_1, \ldots, F_4$ are transfer functions which indicate how abstract states are updated.
Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Control Flow Graph

![Control Flow Graph Image]

Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$\ell_2$</th>
<th>$\ell_3$</th>
<th>$\ell_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>$[0, 0]$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Control Flow Graph

$\ell_1: a = 0;$

$\ell_2: a < 10$

$\ell_3: a++;$

$\ell_4: \ldots$

$\sigma_{\ell_1}$ a $[0, 0] = [0, 0]$
## Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{th}$ loop iter</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>$\perp$</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>[0, 0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Control Flow Graph

1. $\ell_1: a = 0$
2. $\ell_2: a < 10$
3. $\ell_3: a++$
4. $\ell_4: \ldots$

- $a \geq 10$
- $a < 10$

**Update:**
- $\sigma_{\ell_1}(a)$
- $\sigma_{\ell_2}(a)$
- $\sigma_{\ell_3}(a)$

**Intersection:**
$$ a \sqcup \sigma_{\ell_1}(a) \sqcup \sigma_{\ell_3}(a) = [0, 0] $$

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{th}$ loop iter</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[1, 1]$</td>
<td>$[1, 1]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
</tr>
</tbody>
</table>

Control Flow Graph

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>1$^{\text{st}}$ loop iter</th>
<th>2$^{\text{nd}}$ loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>$\bot$</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>[0, 0]</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>

Control Flow Graph:

- $\ell_1: a = 0$
- $\ell_2: a < 10$
- $\ell_3: a++$
- $\ell_4: \ldots$

- $a \geq 10$
- $a < 10$

**Example Calculation**

- $a \sigma_{\ell_1}(a)$
- $a \sigma_{\ell_2}(a)$
- $a ([-\infty, 9] \cap \sigma_{\ell_2}(a)) + [1, 1]$

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{st}$ loop iter</th>
<th>$2^{nd}$ loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[1, 1]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
</tr>
</tbody>
</table>

Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract</th>
<th>Init</th>
<th>After analyzing</th>
<th>1\textsuperscript{st} loop iter</th>
<th>2\textsuperscript{nd} loop iter</th>
<th>...</th>
<th>11\textsuperscript{th} loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace</td>
<td>ℓ\textsubscript{1}</td>
<td>After ℓ\textsubscript{2}</td>
<td>After ℓ\textsubscript{3}</td>
<td>After ℓ\textsubscript{2}</td>
<td>After ℓ\textsubscript{3}</td>
<td>...</td>
</tr>
<tr>
<td>σ\textsubscript{ℓ\textsubscript{1}}(a)</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>...</td>
</tr>
<tr>
<td>σ\textsubscript{ℓ\textsubscript{2}}(a)</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 1]</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>σ\textsubscript{ℓ\textsubscript{3}}(a)</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 2]</td>
</tr>
<tr>
<td>σ\textsubscript{ℓ\textsubscript{4}}(a)</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ l_1: a = 0; \]
\[ a \geq 10 \]
\[ l_2: a < 10 \]
\[ a < 10 \]
\[ l_4: \ldots \]
\[ l_3: a++; \]
\[ a \sigma_{\ell_1}(a) \sqcup \sigma_{\ell_3}(a) \]
\[ a \left( [0, 0] \right) \]
\[ a \left( [-\infty, 9] \cap \sigma_{\ell_2}(a) \right) + [1, 1] \]

Control Flow Graph

### Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing</th>
<th>$1^{st}$ loop iter</th>
<th>$2^{nd}$ loop iter</th>
<th>...</th>
<th>$11^{th}$ loop iter</th>
<th>$12^{nd}$ loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\ell_1$</td>
<td>After $\ell_2$</td>
<td>After $\ell_3$</td>
<td></td>
<td>After $\ell_2$</td>
<td>After $\ell_3$</td>
</tr>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>...</td>
<td>$[0, 0]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 1]$</td>
<td>$[0, 1]$</td>
<td>...</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[1, 1]$</td>
<td>$[1, 1]$</td>
<td>$[1, 2]$</td>
<td>...</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Control Flow Graph

- $\ell_1 : a = 0$
- $\ell_2 : a < 10$
- $\ell_3 : a + +$
- $\ell_4 : \ldots$

<table>
<thead>
<tr>
<th>$\ell_1$</th>
<th>$\sigma_{\ell_1}$</th>
<th>$a$</th>
<th>$[0, 0]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_2$</td>
<td>$\sigma_{\ell_2}$</td>
<td>$\sigma_{\ell_1}(a) \sqcap \sigma_{\ell_3}(a) = [0, 10]$</td>
<td></td>
</tr>
<tr>
<td>$\ell_3$</td>
<td>$\sigma_{\ell_3}$</td>
<td>$a ([-\infty, 9] \sqcap \sigma_{\ell_2}(a)) + [1, 1]$</td>
<td></td>
</tr>
</tbody>
</table>
Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
<th>...</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
<th>...</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>$\bot$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>...</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>...</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 1]$</td>
<td>$[0, 1]$</td>
<td>...</td>
<td>$[0, 10]$</td>
<td>$[0, 10]$</td>
<td>...</td>
<td>$[0, 10]$</td>
<td>$[0, 10]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$[1, 1]$</td>
<td>$[1, 1]$</td>
<td>$[1, 2]$</td>
<td>...</td>
<td>$[1, 10]$</td>
<td>$[1, 10]$</td>
<td>...</td>
<td>$[1, 10]$</td>
<td>$[1, 10]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>...</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>...</td>
<td>$\bot$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>

Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

Abstract trace

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
<th>$\ldots$</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>$\ldots$</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 1]</td>
<td>[0, 1]</td>
<td>$\ldots$</td>
<td>[0, 10]</td>
<td>[0, 10]</td>
<td>[0, 10]</td>
<td>[0, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 2]</td>
<td>$\ldots$</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>$\ldots$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>

Control Flow Graph

Fixpoint is reached!

(Abstract trace after loop round 11 =
Abstract trace after loop round 12)
Abstract Trace: Naive Fixed-Point Computation for Loops

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

Control Flow Graph:

- $\ell_1: a = 0$
- $\ell_2: a < 10$
- $\ell_3: a++$
- $\ell_4: ...$

$\ell_1$ transitions to $\ell_2$ when $a \geq 10$

- $\sigma_{\ell_1}(a) = [0, 0]$
- $\sigma_{\ell_2}(a) = [0, 0]$ ⊕ $\sigma_{\ell_3}(a) = [1, 1]$
- $\sigma_{\ell_4}(a) = (\{\infty, 9\} \cap \sigma_{\ell_2}(a)) + [1, 1] = [10, \infty] \cap \sigma_{\ell_2}(a) = [10, 10]$

Abstract Trace: Naive Fixed-Point Computation for Loops

Control Flow Graph

Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; loop iter</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; loop iter</th>
<th>11&lt;sup&gt;th&lt;/sup&gt; loop iter</th>
<th>12&lt;sup&gt;nd&lt;/sup&gt; loop iter</th>
<th>After analyzing</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 1]$</td>
<td>$[0, 1]$</td>
<td>$[0, 10]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[1, 1]$</td>
<td>$[1, 1]$</td>
<td>$[1, 2]$</td>
<td>$[1, 10]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[10, 10]$</td>
</tr>
</tbody>
</table>

Control Flow Graph:

- $\ell_1$: $a = 0$
- $\ell_2$: $a < 10$
- $\ell_3$: $a +=$
- $\ell_4$: ...

Figures:

- $\sigma_{\ell_1}(a)$: $[0, 0]$
- $\sigma_{\ell_2}(a)$: $[0, 10]$
- $\sigma_{\ell_3}(a)$: $[1, 10]$
- $\sigma_{\ell_4}(a)$: $[10, 10]$

12 iterations while analyzing the loop.

Software Security Analysis 2024
https://github.com/SVF-tools/Software-Security-Analysis
Abstract Trace: Naive Fixed-Point Computation for Loops

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing</th>
<th>1\textsuperscript{st} loop iter</th>
<th>2\textsuperscript{nd} loop iter</th>
<th>11\textsuperscript{th} loop iter</th>
<th>12\textsuperscript{nd} loop iter</th>
<th>After analyzing</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 1]</td>
<td>[0, 1]</td>
<td>[0, 10]</td>
<td>[0, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 2]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[10, 10]</td>
</tr>
</tbody>
</table>

**Control Flow Graph**

- $\ell_1: a = 0$
- $\ell_2: a < 10$
- $\ell_3: a > 10$
- $\ell_4: \ldots$

Initial value: $a = 0$

Iteration 1:
- $\ell_1: a = 0$
- $\ell_2: a < 10$
- $\ell_3: a > 10$
- $\ell_4: \ldots$

Iteration 2:
- $\sigma_{\ell_1}(a) = [0, 0]$
- $\sigma_{\ell_2}(a) = [0, 10]$
- $\sigma_{\ell_3}(a) = [1, 10]$
- $\sigma_{\ell_4}(a) = [10, 10]$

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 $\sigma_{\ell_2}(a)$.

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

\[
[0, 0] \Rightarrow [0, 1] \Rightarrow \ldots \Rightarrow [0, 10] \Rightarrow [0, 10]
\]
Widening technique can accelerate the fixpoint computation of $\sigma_{\ell_2}(a)$.

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

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

**Widening**

generously update $\sigma_{\ell_2}(a)$

$$[0, +\infty]$$
Widening: Accelerating Fixed-Point Computation

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

\[
\ell_1: a = 0; \quad l_1
\]
\[
\ell_2: a < 10 \quad l_2
\]
\[
\ell_3: a++; \quad l_3
\]
\[
\ell_4:... \quad l_4
\]

Control Flow Graph

\[\sigma_{l_2}(a) := \sigma_{l_1}(a) \cup \sigma_{l_3}(a)\]

Apply widening operator $\nabla$

\[\sigma^k_{l_2}(a) := \sigma^k_{l_2}(a) \nabla (\sigma_{l_1}(a) \cup \sigma^k_{l_3}(a))\]

$\sigma_{l_2}$ denotes the value of $\sigma_{l_2}$ after the $k^{th}$ analysis of $l_2$, and $\sigma_{l_1}$ does not have a superscription as it is updated only once and is not involved in the loop.

https://github.com/SVF-tools/Software-Security-Analysis
Widening: Accelerating Fixed-Point Computation

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

\[
\ell_1: a = 0; \\
\ell_2: a < 10 \\
\ell_3: a++; \\
\ell_4: \ldots
\]

Control Flow Graph

\[
\sigma_{\ell_2}(a) := \sigma_{\ell_1}(a) \cup \sigma_{\ell_3}(a)
\]

\[
\sigma^k_{\ell_2}(a) := \sigma^k_{\ell_2}(a) \cap \sigma^k_{\ell_1}(a) \cup \sigma^k_{\ell_3}(a)
\]

Apply widening operator $\nabla$

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

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

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

As a concrete example, $[0, 0] \triangledown [0, 1] = [0, +\infty]$. 
Widening Operator

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

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

where

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

As a concrete example, $[0, 0] \nabla [0, 1] = [0, +\infty]$. 
## Widening: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After $\ell_1$</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
<th>After $\ell_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Control Flow Graph

- $\ell_1: a = 0$
- $a \geq 10 \Rightarrow \ell_2: a < 10$
- $a < 10 \Rightarrow \ell_3: a++$
- $\ell_4: \ldots$

#### Control Flow Graph
Widening: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$\ell_2$</th>
<th>$\ell_3$</th>
<th>$\ell_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>\bot</td>
<td>[0, 0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>\bot</td>
<td>\bot</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>\bot</td>
<td>\bot</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>\bot</td>
<td>\bot</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Control Flow Graph

$\ell_1: a = 0$

$\ell_2: a < 10$

$\ell_4: \ldots$

$\ell_3: a++$

$\sigma_{\ell_1}$ $\bot$ \[0, 0\] $\bot$ $\bot$ $\bot$

$\sigma_{\ell_2}$ $\bot$ $\bot$ $\bot$ $\bot$ $\bot$

$\sigma_{\ell_3}$ $\bot$ $\bot$ $\bot$ $\bot$ $\bot$

$\sigma_{\ell_4}$ $\bot$ $\bot$ $\bot$ $\bot$ $\bot$

Control Flow Graph

Software Security Analysis 2024

https://github.com/SVF-tools/Software-Security-Analysis
# Widening: The Loop Example

## Abstract

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{st}$ loop iter</th>
<th>After $\ell_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td></td>
</tr>
</tbody>
</table>

## Control Flow Graph

- $\ell_1: a = 0$
- $\ell_2: a < 10$
- $\ell_3: a += 1$
- $\ell_4: ...$

### Example

- $a \geq 10$
  - $\ell_2: a < 10$
  - $\ell_3: a += 1$

### Widening

$a[0, 0]$

$\sigma_{\ell_1}(a) \sqcap \sigma_{\ell_2}(a) = [0, 0]$
Widening: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{th}$ loop iter</th>
<th>$2^{nd}$ loop iter</th>
<th>$3^{rd}$ loop iter</th>
<th>$4^{th}$ loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \sigma_{\ell_1}(a) \perp [0, 0], [0, 0], [0, 0], [0, 0] \]
\[ \sigma_{\ell_2}(a) \perp [0, 0], [0, 0], [1, 1], [0, \infty] \]
\[ \sigma_{\ell_3}(a) \perp [1, 1], [1, 10], [1, 1], [1, 1] \]

Control Flow Graph

$\ell_1: a = 0$

$\ell_2: a < 10$

$\ell_3: a + = $
Widening: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{st}$ loop iter</th>
<th>$2^{nd}$ loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[1, 1]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
</tr>
</tbody>
</table>

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

\[ \ell_1 : a = 0; \]
\[ \ell_1 \]
\[ a \geq 10 \]
\[ \ell_2 : a < 10 \]
\[ \ell_2 \]
\[ a < 10 \]
\[ \ell_3 : a++; \]
\[ \ell_3 \]
\[ \ell_4 : \ldots \]

\[ a \]
\[ 0, 0 \]

\[ \sigma_{\ell_1} \]

\[ a \]

\[ \sigma_{\ell_2}^2 \]

\[ a \]

\[ \sigma_{\ell_3}^1(a) \bigvee (\sigma_{\ell_1}^1(a) \sqcup \sigma_{\ell_3}^1(a)) = [0, \infty] \]
Widening: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>1$^{st}$ loop iter</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
<th>2$^{nd}$ loop iter</th>
<th>After $\ell_2$</th>
<th>After $\ell_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>

Control Flow Graph:

- $\ell_1: a = 0$
- $\ell_2: a < 10$
- $\ell_3: a +=$
- $\ell_4: \ldots$

$\sigma_{\ell_1}(a) \triangleright [0, 0]$

$\sigma_{\ell_2}(a) \triangleright (\sigma_{\ell_1}(a) \triangleright \sigma_{\ell_3}(a)) \triangleright [1, 10]$

$\sigma_{\ell_3}(a) \triangleright ([-\infty, 9] \cap \sigma_{\ell_2}(a)) + [1, 1] = [1, 10]$

### Abstract

After analyzing $\ell_1$, the abstract trace $\sigma_{\ell_1}(a)$ is initialized as $\perp$. For the first loop iteration, the trace transitions to $[0, 0]$. For the second loop iteration, the trace transitions to $[0, 0]$. For the third loop iteration, the trace transitions to $[0, 0]$.

### Control Flow Graph

- **$\ell_1$:** $a = 0$; $a \geq 10$
- **$\ell_2$:** $a < 10$
- **$\ell_3$:** $a += 1$
- **$\ell_4$:**...

### After Analyzing

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>1$^{st}$ loop iter</th>
<th>2$^{nd}$ loop iter</th>
<th>3$^{rd}$ loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, \infty]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$[1, 1]$</td>
<td>$[1, 10]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
</tr>
</tbody>
</table>

### Example

- $a$ $\sigma_{\ell_1}$ $[0, 0]$
- $a$ $\sigma_{\ell_2}$ $[0, \infty]$
- $a$ $\sigma_{\ell_3}$ $[1, 10]$
- $a$ $\sigma_{\ell_4}$ $[1, 10]$

### Software Security Analysis 2024

https://github.com/SVF-tools/Software-Security-Analysis
Widening: The Loop Example

Abstract trace | Init | After analyzing $\ell_1$ | 1$^{st}$ loop iter | 2$^{nd}$ loop iter | 3$^{rd}$ loop iter
---|---|---|---|---|---
$\sigma_{\ell_1}(a)$ | ⊥ | [0, 0] | [0, 0] | [0, 0] | [0, 0] | [0, 0]
$\sigma_{\ell_2}(a)$ | ⊥ | ⊥ | [0, 0] | [0, 0] | [0, 0] | [0, 0] | [0, 0]
$\sigma_{\ell_3}(a)$ | ⊥ | ⊥ | ⊥ | [1, 1] | [1, 1] | [1, 10] | [1, 10]
$\sigma_{\ell_4}(a)$ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥ | ⊥

Control Flow Graph

## Widening: The Loop Example

### Abstract

After analyzing $\ell_1$, the result is $[0, 0]$. After analyzing $\ell_2$, the result is $[0, 0]$. After analyzing $\ell_3$, the result is $[0, 0]$. After analyzing $\ell_2$, the result is $[0, 0]$. After analyzing $\ell_3$, the result is $[0, 0]$.

### Control Flow Graph

- $\ell_1$: $a = 0$
- $\ell_2$: $a < 10$
- $\ell_3$: $a = 10$
- $\ell_4$: $\ldots$

### Fixpoint is reached!

(Abstract trace after loop round 2 = Abstract trace after loop round 3)

### Software Security Analysis 2024

https://github.com/SVF-tools/Software-Security-Analysis
## Widening: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{st}$ loop iter</th>
<th>$2^{nd}$ loop iter</th>
<th>$3^{rd}$ loop iter</th>
<th>After analyzing $\ell_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[10, ∞]</td>
</tr>
</tbody>
</table>

### Control Flow Graph

- $\ell_1: a = 0$
- $\ell_2: a < 10$
- $\ell_3: a = ++$
- $\ell_4: \ldots$

### Equations

- $a \geq 10$
- $a < 10$

- $a[0, 0]$
- $\sigma_{\ell_1}(a)$
- $\sigma_{\ell_2}(a) \nabla (\sigma_{\ell_1}(a) \sqcup \sigma_{\ell_3}(a))$
- $a([\infty, 9] \cap \sigma_{\ell_2}(a)) + [1, 1]$
- $a[10, \infty] \cap \sigma_{\ell_2}(a) = [10, \infty]$
Widening: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing</th>
<th>1\textsuperscript{st} loop iter</th>
<th>2\textsuperscript{nd} loop iter</th>
<th>3\textsuperscript{rd} loop iter</th>
<th>After analyzing</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
<td>$[0, 0]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>$[0, 0]$</td>
<td>$[0, \infty]$</td>
<td>$[0, \infty]$</td>
<td>$[0, \infty]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>$[1, 1]$</td>
<td>$[1, 10]$</td>
<td>$[1, 10]$</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>

Control Flow Graph

- $\ell_1: a = 0$
- $\ell_2: a < 10$
- $\ell_3: a++;$
- $\ell_4: \ldots$

After analyzing

- $\sigma_{\ell_1}: a \geq 10$
- $\sigma_{\ell_2}: a < 10$
- $\sigma_{\ell_3}: a
- $\sigma_{\ell_4}: a$

## Widening: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{st}$ loop iter</th>
<th>$2^{nd}$ loop iter</th>
<th>$3^{rd}$ loop iter</th>
<th>$4^{th}$ loop iter</th>
<th>After analyzing $\ell_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, ∞]</td>
<td>[0, ∞]</td>
<td>[0, ∞]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[10, ∞]</td>
</tr>
</tbody>
</table>

### Control Flow Graph

- $\ell_1: a = 0$
- $\ell_2: a < 10$
- $\ell_3: a + a$
- $\ell_4: ...$

### Widening Example

- $\sigma_{\ell_1}(a) \perp [0, 0] \Rightarrow [0, 0] \Rightarrow [0, ∞]$  
- $\sigma_{\ell_2}(a) \perp \perp [0, 0] \Rightarrow [0, 0] \Rightarrow [0, ∞] \Rightarrow [0, ∞]$  
- $\sigma_{\ell_3}(a) \perp \perp \perp [1, 10] \Rightarrow [1, 10] \Rightarrow [1, 10] \Rightarrow [1, 10]$  
- $\sigma_{\ell_4}(a) \perp \perp \perp \perp \perp \perp \perp \perp \perp \perp [10, ∞]$
### Abstract

After analyzing the loop, the result is consistent across iterations.

### Table: Widening Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{st}$ loop iter</th>
<th>$2^{nd}$ loop iter</th>
<th>$3^{rd}$ loop iter</th>
<th>After analyzing $\ell_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, ∞]</td>
<td>[0, ∞]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[10, ∞]</td>
</tr>
</tbody>
</table>

### Diagram: Control Flow Graph

- $\ell_1: a = 0$
- $\ell_2: a < 10$
- $\ell_3: a++$
- $\ell_4: \ldots$

After analyzing the loop:

- $[0, 0] \Rightarrow [0, ∞] \Rightarrow [0, ∞]$

**3 iterations** while analyzing the loop.

Faster than naive fixpoint computation (12 iterations)!
Widening: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing</th>
<th>After</th>
<th>After</th>
<th>After</th>
<th>After</th>
<th>After</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\ell_1$</td>
<td>$\ell_2$</td>
<td>$\ell_3$</td>
<td>$\ell_2$</td>
<td>$\ell_3$</td>
<td>$\ell_2$</td>
<td>$\ell_3$</td>
</tr>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, ∞]</td>
<td>[0, ∞]</td>
<td>[0, ∞]</td>
<td>[0, ∞]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[10, ∞]</td>
</tr>
</tbody>
</table>

Control Flow Graph

$\ell_1: a = 0$; $\ell_2: a < 10$; $\ell_3: a++;$ $\ell_4: \ldots$

$\sigma_{\ell_1} a [0, 0] \Rightarrow \sigma_{\ell_2} a [0, \infty] \Rightarrow [0, 10]$

Less precise than without widening

Faster than without widening (12 iterations)!

3 iterations while analyzing the loop

$[0, 0] \Leftrightarrow [0, \infty] \Leftrightarrow [0, \infty]$
Narrowing: Precision Refinement

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

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

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

Widening

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

[0, $+\infty$]
Narrowing: Precision Refinement

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

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

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

Widening: aggressively update $\sigma_{\ell_2}(a)$

Narrowing: conservatively update $\sigma_{\ell_2}(a)$
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 $\sigma_{\ell_2}$.

\[
\sigma_{\ell_2}^k(a) := \sigma_{\ell_2}^{k-1}(a) \triangledown (\sigma_{\ell_1}(a) \sqcup \sigma_{\ell_3}^{k-1}(a))
\]
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 $\sigma_{\ell_2}$.

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

Apply narrowing operator $\Delta$ instead

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

What is a Narrowing Operator?

---

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
The Narrowing Operator \( \Delta : \mathbb{A} \times \mathbb{A} \rightarrow \mathbb{A} \) is formally defined on a poset \( (\mathbb{A}, \sqsubseteq) \). \( \Delta \) on interval domain could be defined as:

\[
[l_1, h_1] \Delta [l_2, h_2] = [l_3, h_3]
\]
The Narrowing Operator $\left( \Delta : A \times A \rightarrow A \right)$ is formally defined on a poset $(A, \sqsubseteq)$. $\Delta$ on interval domain could be defined as:

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

where

\[
\begin{align*}
\begin{array}{c}
l_3 = \left\{ \\
\begin{array}{l}
l_2 \quad l_1 \equiv -\infty \\
l_1 \quad l_1 \neq -\infty
\end{array}
\end{array},
\begin{array}{c}
h_3 = \left\{ \\
\begin{array}{l}
h_2 \quad h_1 \equiv \infty \\
h_1 \quad h_1 \neq \infty
\end{array}
\end{array}
\end{align*}
\]

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

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
Narrowing: The Loop Example

Abstract trace

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing</th>
<th>1st loop iter</th>
<th>2nd loop iter</th>
<th>3rd loop iter</th>
<th>4th loop iter</th>
<th>5th loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>After</td>
<td>After</td>
<td>After</td>
<td>After</td>
<td>After</td>
<td>After</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ℓ₁</td>
<td>ℓ₂</td>
<td>ℓ₂</td>
<td>ℓ₂</td>
<td>ℓ₃</td>
<td>ℓ₃</td>
</tr>
<tr>
<td>σₗ₁(a)</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>σₗ₂(a)</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, ∞]</td>
<td>[0, ∞]</td>
<td>[0, ∞]</td>
<td>[0, ∞]</td>
</tr>
<tr>
<td>σₗ₃(a)</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>σₗ₄(a)</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>

Widening reaches a fixpoint at the 3rd loop iteration

\[
\sigma^{3}_{\ell_2} (a) \nabla (\sigma^{1}_{\ell_1} (a) \sqcup \sigma^{2}_{\ell_3} (a)) \]

\[
\sigma^{3}_{\ell_3} (a) = [\langle -\infty, 9 \rangle \cap \sigma^{3}_{\ell_2} (a)] + [1, 1] \]

Control Flow Graph

## Narrowing: The Loop Example

### Abstract

After analyzing $\ell_1$th loop iter

<table>
<thead>
<tr>
<th>$\ell_2$</th>
<th>$\ell_3$</th>
<th>$\ell_2$</th>
<th>$\ell_3$</th>
<th>$\ell_2$</th>
<th>$\ell_3$</th>
<th>$\ell_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
</tbody>
</table>

### $\sigma_{\ell_1}(a)$

<table>
<thead>
<tr>
<th>$\ell_2$</th>
<th>$\ell_3$</th>
<th>$\ell_2$</th>
<th>$\ell_3$</th>
<th>$\ell_2$</th>
<th>$\ell_3$</th>
<th>$\ell_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
</tbody>
</table>

### $\sigma_{\ell_2}(a)$

<table>
<thead>
<tr>
<th>$\ell_3$</th>
<th>$\ell_3$</th>
<th>$\ell_2$</th>
<th>$\ell_3$</th>
<th>$\ell_2$</th>
<th>$\ell_3$</th>
<th>$\ell_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
</tbody>
</table>

### Control Flow Graph

- **Start narrowing at the 4th loop iteration**

#### Loop Iterations

1. $\ell_1: a = 0$
   - After $\ell_1$, $\sigma_{\ell_1}(a) = [0, 0]$

2. $\ell_2: a < 10$
   - After $\ell_2$, $\sigma_{\ell_2}(a) = [0, 0]$

3. $\ell_3: a = a + 1$
   - After $\ell_3$, $\sigma_{\ell_3}(a) = [1, 10]$

4. $\ell_4: \ldots$

#### Narrowing Equations

- $a \geq 10$
  - $\sigma_{\ell_2}(a) \Delta (\sigma_{\ell_1}(a) \sqcup \sigma_{\ell_3}(a)) = [0, 10]$

- $a < 10$
  - $a [\neg \infty, 9) \sqcap \sigma_{\ell_3}(a) = [1, 1]$

---

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
# Narrowing: The Loop Example

## Abstract

After analyzing, we trace the behavior of the loop through each iteration. Below is a table summarizing the narrowing of the data flow from before to after analyzing the loop.

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{st}$ loop iter</th>
<th>$2^{nd}$ loop iter</th>
<th>$3^{rd}$ loop iter</th>
<th>$4^{th}$ loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, ∞]</td>
<td>[0, ∞]</td>
<td>[0, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>

### Control Flow Graph

- $\ell_1 : a = 0$;  
- $\ell_2 : a < 10$;  
- $\ell_3 : a++;$  
- $\ell_4 : \ldots$

### Examples

- $\sigma_{\ell_1}(a)$  
  - $a \geq 10$  
  - $a < 10$

- $\sigma_{\ell_2}(a)$  
  - $\sigma_{\ell_2}(a) \triangle (\sigma_{\ell_1}(a) \sqcup \sigma_{\ell_3}^3(a))$

- $\sigma_{\ell_3}(a)$  
  - $([-\infty, 9] \cap \sigma_{\ell_2}(a)) + [1, 1] = [1, 10]$

Software Security Analysis 2024  
Narrowing: The Loop Example

Abstract trace

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; loop iter</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; loop iter</th>
<th>3&lt;sup&gt;rd&lt;/sup&gt; loop iter</th>
<th>4&lt;sup&gt;th&lt;/sup&gt; loop iter</th>
<th>5&lt;sup&gt;th&lt;/sup&gt; loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>After (\ell_2)</td>
<td>After (\ell_3)</td>
<td>After (\ell_2)</td>
<td>After (\ell_3)</td>
<td>After (\ell_2)</td>
</tr>
<tr>
<td>(\sigma_{\ell_1}(a))</td>
<td>(\perp)</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>(\sigma_{\ell_2}(a))</td>
<td>(\perp)</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, (\infty)]</td>
<td>[0, (\infty)]</td>
<td>[0, 10]</td>
<td>[0, 10]</td>
</tr>
<tr>
<td>(\sigma_{\ell_3}(a))</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>(\sigma_{\ell_4}(a))</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
</tr>
</tbody>
</table>

Control Flow Graph

\(\ell_1: a = 0;\)

\(\ell_2: a < 10\)

\(\ell_3: a++;\)

\(\ell_4: \ldots\)

Software Security Analysis 2024

https://github.com/SVF-tools/Software-Security-Analysis
## Narrowing: The Loop Example

### Abstract

After analyzing \( \ell_1 \), \( \ell_2 \), \( \ell_3 \), \( \ell_4 \), the domains for each loop iteration are as follows:

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing</th>
<th>1(^{st}) loop iter</th>
<th>2(^{nd}) loop iter</th>
<th>3(^{rd}) loop iter</th>
<th>4(^{th}) loop iter</th>
<th>5(^{th}) loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_{\ell_1}(a) )</td>
<td>\perp</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
</tr>
<tr>
<td>( \sigma_{\ell_2}(a) )</td>
<td>\perp</td>
<td>\perp</td>
<td>[0,0]</td>
<td>[0,\infty]</td>
<td>[0,\infty]</td>
<td>[0,10]</td>
<td>[0,10]</td>
</tr>
<tr>
<td>( \sigma_{\ell_3}(a) )</td>
<td>\perp</td>
<td>\perp</td>
<td>\perp</td>
<td>[1,1]</td>
<td>[1,10]</td>
<td>[1,10]</td>
<td>[1,10]</td>
</tr>
<tr>
<td>( \sigma_{\ell_4}(a) )</td>
<td>\perp</td>
<td>\perp</td>
<td>\perp</td>
<td>\perp</td>
<td>\perp</td>
<td>\perp</td>
<td>\perp</td>
</tr>
</tbody>
</table>

### Control Flow Graph

1. \( \ell_1: a = 0; \)
2. \( \ell_2: a < 10 \)
3. \( \ell_3: a++; \)
4. \( \ell_4: \ldots \)

#### Domain Analysis

- \( \sigma_{\ell_1}(a) \): \([0,0]\)
- \( \sigma_{\ell_2}(a) \): \([0,\infty]\)
- \( \sigma_{\ell_3}(a) \): \([1,10]\)
- \( \sigma_{\ell_4}(a) \): Domain not shown

#### Domain Calculations

- \( \sigma_{\ell_2}(a) \Delta (\sigma_{\ell_1}(a) \sqcap \sigma_{\ell_3}^4(a)) \)
- \( \sigma_{\ell_3}(a) \) \([\infty, 9] \sqcap \sigma_{\ell_2}^5(a) + [1, 1] = [1,10] \)

---

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
Narrowing: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing</th>
<th>1\text{st} loop iter</th>
<th>2\text{nd} loop iter</th>
<th>3\text{rd} loop iter</th>
<th>4\text{th} loop iter</th>
<th>5\text{th} loop iter</th>
<th>After analyzing</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0,0]</td>
<td>[0,0]</td>
<td>[0,∞]</td>
<td>[0,∞]</td>
<td>[0,10]</td>
<td>[0,10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1,1]</td>
<td>[1,1]</td>
<td>[1,10]</td>
<td>[1,10]</td>
<td>[1,10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
</tbody>
</table>

Control Flow Graph

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

https://github.com/SVF-tools/Software-Security-Analysis
Narrowing: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{th}$ loop iter</th>
<th>$2^{nd}$ loop iter</th>
<th>$3^{rd}$ loop iter</th>
<th>$4^{th}$ loop iter</th>
<th>$5^{th}$ loop iter</th>
<th>After analyzing $\ell_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>1</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>1</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
</tbody>
</table>

Control Flow Graph:

- $\ell_1: a = 0$
- $\ell_2: a < 10$
- $\ell_3: a ++$
- $\ell_4: ...$

After analyzing $\ell_1$:

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

After analyzing $\ell_2$:

- $\sigma_{\ell_2}(a) = ([0, 0])$

After analyzing $\ell_3$:

- $\sigma_{\ell_3}(a) = [1, 10]$

After analyzing $\ell_4$:

- $\sigma_{\ell_4}(a) = [10, 10]$

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
Narrowing: The Loop Example

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{st}$ loop iter</th>
<th>$2^{nd}$ loop iter</th>
<th>$3^{rd}$ loop iter</th>
<th>$4^{th}$ loop iter</th>
<th>$5^{th}$ loop iter</th>
<th>After analyzing $\ell_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[10, 10]</td>
</tr>
</tbody>
</table>

Control Flow Graph

Software Security Analysis 2024  
https://github.com/SVF-tools/Software-Security-Analysis
Narrowing: The Loop Example

### Abstract
After analyzing $\ell_1$, $\ell_2$, $\ell_3$, and $\ell_4$, the control flow graph shows the evolution of the variable $a$ through the loop iterations.

### Table: Abstract Trace
<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>After analyzing $\ell_2$</th>
<th>After analyzing $\ell_3$</th>
<th>After analyzing $\ell_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, ∞]</td>
<td>[0, ∞]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[10, 10]</td>
</tr>
</tbody>
</table>

### Diagram
- **$\ell_1$:** $a = 0$
- **$\ell_2$:** $a < 10$
- **$\ell_3$:** $a++$
- **$\ell_4$:** ...

### 5 iterations while analyzing the loop
- $[0, 0] \Rightarrow [0, ∞] \Rightarrow [0, ∞] \Rightarrow [0, 10] \Rightarrow [0, 10]$
## Narrowing: The Loop Example

### Control Flow Graph

<table>
<thead>
<tr>
<th>Abstract trace</th>
<th>Init</th>
<th>After analyzing $\ell_1$</th>
<th>$1^{\text{st}}$ loop iter</th>
<th>$2^{\text{nd}}$ loop iter</th>
<th>$3^{\text{rd}}$ loop iter</th>
<th>$4^{\text{th}}$ loop iter</th>
<th>$5^{\text{th}}$ loop iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{\ell_1}(a)$</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>$\sigma_{\ell_2}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>[0, 0]</td>
<td>[0, $\infty$]</td>
<td>[0, $\infty$]</td>
<td>[0, 10]</td>
<td>[0, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_3}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>$\sigma_{\ell_4}(a)$</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>[10, 10]</td>
</tr>
</tbody>
</table>

### Software Security Analysis 2024
https://github.com/SVF-tools/Software-Security-Analysis

---

**5 iterations** while analyzing the loop

- Faster!
- Precise!