Introduction to Program Analysis

15. Static Analysis by Monotonic Closure

Kihong Heo

KAIST

Constraint-based Analysis

- A specialized framework: static analysis by monotonic closure
- Static analysis = setting up initial facts + collecting new facts by rules
- Assumption 1: there are only a finite number of facts
  - E.g., Points-to, dynamic jump targets
- Assumption 2: the abstract domain is the power set of facts
  - E.g., Set of pointers, set of target addresses

Inference Rules

- Methods to **derive new facts from known facts** (i.e., derivable = provable)
- Notation of inference rule:

$$\frac{P_1 \quad \cdots \quad P_n}{P} \quad \begin{array}{l}\textit{premise(s)} \\ \textit{conclusion}\end{array}$$

$$\frac{human(x) \implies mortal(x) \qquad human(s)}{mortal(s)}$$

- Inference rule with zero premise is called an **axiom**

- Examples: inference rules of propositional logic

$$\frac{}{\text{true}} \quad \frac{A \qquad B}{A \land B} \quad \frac{A \land B}{A} \quad \frac{A \land B}{B} \quad \frac{A}{A \lor B} \quad \frac{B}{A \lor B} \quad \frac{A \qquad A \implies B}{B} \quad \ldots$$

Case Study 1: Pointer Analysis

- Goal: the set of locations that each pointer may store globally
  - Computing "points-to" facts between two variables
  - $a \rightarrow b$: "a points to b"
- Example:

$$
\begin{array}{lcl}
C & \to & L := R \\
 & \mid & C ; C \\
 & \mid & \texttt{while}\ B\ C \\
L & \to & x \mid *x \\
R & \to & x \mid *x \mid \&x \\
B & & \text{boolean expressions}
\end{array}
$$

```
x = &a;
y = &x;
while (*) {
  *y = &b;
}
*x = *y
```

All possible points-to facts

{x→a, y→x, x→b, a→a, b→b, a→b, b→a}

Rules for Pointer Analysis

- The rule has the following form $$\frac{C \quad i_1 \cdots i_k}{j}$$
    - Premises: the program has component C and the current set has $i_1 \cdots i_k$
    - Conclusion: j is derivable (so, add j to the set)

Base case (collecting initial facts)

$$\frac{x := \&y}{x \rightarrow y}$$

Inductive cases (deriving new facts)

$$\frac{x := y \quad y \rightarrow z}{x \rightarrow z} \qquad \frac{*x := y \quad x \rightarrow w \quad y \rightarrow z}{w \rightarrow z} \qquad \frac{*x := \&y \quad x \rightarrow w}{w \rightarrow y}$$

$$\frac{x := *y \quad y \rightarrow z \quad z \rightarrow w}{x \rightarrow w} \qquad \frac{*x := *y \quad x \rightarrow w \quad y \rightarrow z \quad z \rightarrow v}{w \rightarrow v}$$

Closure

- Static analysis result = a closure of rules = a fixed point of rules
- Fixed point iteration:
  1. Start from the initial facts
  2. Apply rules and collect more facts
  3. Repeat until no more new facts are generated

Example

```
x = &a;
y = &x;
while (*) {
  *y = &b;
}
*x = *y
```

$$\frac{x := \&y}{x \rightarrow y}$$

$$\frac{x := y \quad y \rightarrow z}{x \rightarrow z} \qquad \frac{*x := y \quad x \rightarrow w \quad y \rightarrow z}{w \rightarrow z} \qquad \frac{*x := \&y \quad x \rightarrow w}{w \rightarrow y}$$

$$\frac{x := *y \quad y \rightarrow z \quad z \rightarrow w}{x \rightarrow w} \qquad \frac{*x := *y \quad x \rightarrow w \quad y \rightarrow z \quad z \rightarrow v}{w \rightarrow v}$$

Collect the initial facts directly from program text

{x→a, y→x}

Example

```
x = &a;
y = &x;
while (*) {
  *y = &b;
}
*x = *y
```

$$\frac{x := \&y}{x \rightarrow y} \qquad \frac{x := y \quad y \rightarrow z}{x \rightarrow z} \qquad \frac{*x := y \quad x \rightarrow w \quad y \rightarrow z}{w \rightarrow z} \qquad \boxed{\frac{*x := \&y \quad x \rightarrow w}{w \rightarrow y}}$$

$$\frac{x := *y \quad y \rightarrow z \quad z \rightarrow w}{x \rightarrow w} \qquad \frac{*x := *y \quad x \rightarrow w \quad y \rightarrow z \quad z \rightarrow v}{w \rightarrow v}$$

Apply the rules to the known facts

{x→a, y→x, x→b}

Example

```
x = &a;
y = &x;
while (*) {
  *y = &b;
}
*x = *y
```

$$\frac{x := \&y}{x \to y}$$

$$\frac{x := y \quad y \to z}{x \to z} \qquad \frac{*x := y \quad x \to w \quad y \to z}{w \to z} \qquad \frac{*x := \&y \quad x \to w}{w \to y}$$

$$\frac{x := *y \quad y \to z \quad z \to w}{x \to w} \qquad \boxed{\frac{*x := *y \quad x \to w \quad y \to z \quad z \to v}{w \to v}}$$

Repeat until reaching a fixed point

{x→a, y→x, x→b, a→a, b→b, a→b, b→a}

Case Study 2: Taint Analysis

- Goal: find an unsanitized information flow from a source point to a sink point
  - Source: read untrustworthy inputs (taint)
  - Sanitizer: neutralize malicious inputs
  - Sink: safety- / security-critical points
- Compute def-use facts between two variables
  - $a \rightarrow b$: "the value of a is used to define the value of b w/o sanitization"
- Applications: format string attack, code injection attack, etc

Rules for Taint Analysis

Base cases
(collecting initial facts)

$$\frac{l : x := E}{\mathsf{Def}(l, x)} \quad \frac{l : E := y}{\mathsf{Use}(l, y)} \quad \frac{l : x := \texttt{source}()}{\mathsf{Src}(l)} \quad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{San}(l)} \quad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{Use}(l, y)} \quad \frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Sink}(l)} \quad \frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Use}(l, x)}$$

Inductive cases
(deriving new facts)

$$\frac{\mathsf{Def}(l_1, x) \quad \mathsf{Use}(l_2, x)}{\mathsf{Edge}(l_1, l_2)} \quad \frac{\mathsf{Src}(l_1) \quad \mathsf{Edge}(l_1, l_2)}{\mathsf{Path}(l_1, l_2)} \quad \frac{\mathsf{Path}(l_1, l_2) \quad \neg\mathsf{San}(l_2) \quad \mathsf{Edge}(l_2, l_3)}{\mathsf{Path}(l_1, l_3)} \quad \frac{\mathsf{Path}(l_1, l_2) \quad \mathsf{Sink}(l_2)}{\mathsf{Alarm}(l_1, l_2)}$$

Example

```
1: x = source();
2: y = x;
3: if(*) {
4:   z = sanitize(y);
5: } else {
6:   sink(y);
7: }
8: sink(z);
```

$$\frac{l : x := E}{\mathsf{Def}(l, x)} \quad \frac{l : E := y}{\mathsf{Use}(l, y)} \quad \frac{l : x := \texttt{source}()}{\mathsf{Src}(l)}$$

$$\frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{San}(l)} \quad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{Use}(l, y)}$$

$$\frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Sink}(l)} \quad \frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Use}(l, x)}$$

Collect the initial facts directly from program text

{ $\mathsf{Def}(1,x)$, $\mathsf{Src}(1,x)$, $\mathsf{Def}(2,y)$, $\mathsf{Use}(2,x)$ **}**

Example

```
1: x = source();
2: y = x;
3: if(*) {
4:   z = sanitize(y);
5: } else {
6:   sink(y);
7: }
8: sink(z);
```

$$\frac{l : x := E}{\mathsf{Def}(l, x)} \quad \frac{l : E := y}{\mathsf{Use}(l, y)} \quad \frac{l : x := \texttt{source}()}{\mathsf{Src}(l)}$$

$$\frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{San}(l)} \quad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{Use}(l, y)}$$

$$\frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Sink}(l)} \quad \frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Use}(l, x)}$$

Collect the initial facts directly from program text

{ Def$(1,x)$, Src$(1,x)$, Def$(2,y)$, Use$(2,x)$,
Def$(4,z)$, Use$(4,y)$, San(4) }

Example

```
1: x = source();
2: y = x;
3: if(*) {
4:   z = sanitize(y);
5: } else {
6:   sink(y);
7: }
8: sink(z);
```

$$\frac{l : x := E}{\mathsf{Def}(l, x)} \quad \frac{l : E := y}{\mathsf{Use}(l, y)} \quad \frac{l : x := \texttt{source}()}{\mathsf{Src}(l)}$$

$$\frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{San}(l)} \quad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\mathsf{Use}(l, y)}$$

$$\boxed{\frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Sink}(l)} \quad \frac{l : \texttt{sink}(\texttt{x})}{\mathsf{Use}(l, x)}}$$

Collect the initial facts directly from program text

{ Def$(1,x)$, Src(1), Def$(2,y)$, Use$(2,x)$,
Def$(4,z)$, Use$(4,y)$, San(4),
Sink(6), Use$(6,y)$ }

Example

```
1: x = source();
2: y = x;
3: if(*) {
4:   z = sanitize(y);
5: } else {
6:   sink(y);
7: }
8: sink(z);
```

$$\frac{l : x := E}{\textsf{Def}(l, x)} \qquad \frac{l : E := y}{\textsf{Use}(l, y)} \qquad \frac{l : x := \texttt{source}()}{\textsf{Src}(l)}$$

$$\frac{l : x := \texttt{sanitize}(\texttt{y})}{\textsf{San}(l)} \qquad \frac{l : x := \texttt{sanitize}(\texttt{y})}{\textsf{Use}(l, y)}$$

$$\boxed{\frac{l : \texttt{sink}(\texttt{x})}{\textsf{Sink}(l)} \qquad \frac{l : \texttt{sink}(\texttt{x})}{\textsf{Use}(l, x)}}$$

Collect the initial facts directly from program text

{ Def$(1,x)$, Src(1), Def$(2,y)$, Use$(2,x)$,
Def$(4,z)$, Use$(4,y)$, San(4),
Sink(6), Use$(6,y)$,
Sink(8), Use$(8,z)$}

Example

```
x = source();
y = x;
if(*) {
  z = sanitize(y);
} else {
  sink(y);
}
sink(z);
```

$$\boxed{\frac{\mathsf{Def}(l_1, x) \quad \mathsf{Use}(l_2, x)}{\mathsf{Edge}(l_1, l_2)}} \qquad \frac{\mathsf{Src}(l_1) \quad \mathsf{Edge}(l_1, l_2)}{\mathsf{Path}(l_1, l_2)} \qquad \frac{\mathsf{Path}(l_1, l_2) \quad \neg\mathsf{San}(l_2) \quad \mathsf{Edge}(l_2, l_3)}{\mathsf{Path}(l_1, l_3)}$$

$$\frac{\mathsf{Path}(l_1, l_2) \quad \mathsf{Sink}(l_2)}{\mathsf{Alarm}(l_1, l_2)}$$

Apply the rules to the known facts

{ Def$(1,x)$, Src(1), Def$(2,y)$, Use$(2,x)$,
Def$(4,z)$, Use$(4,y)$, San(4),
Sink(6), Use$(6,y)$,
Sink(8), Use$(8,z)$,
Edge$(1,2)$, Edge$(2,4)$, Edge$(2,6)$}

Example

```
1: x = source();
2: y = x;
3: if(*) {
4:   z = sanitize(y);
5: } else {
6:   sink(y);
7: }
8: sink(z);
```

$$\frac{\mathsf{Def}(l_1, x) \quad \mathsf{Use}(l_2, x)}{\mathsf{Edge}(l_1, l_2)} \qquad \frac{\mathsf{Src}(l_1) \quad \mathsf{Edge}(l_1, l_2)}{\mathsf{Path}(l_1, l_2)} \qquad \frac{\mathsf{Path}(l_1, l_2) \quad \neg\mathsf{San}(l_2) \quad \mathsf{Edge}(l_2, l_3)}{\mathsf{Path}(l_1, l_3)}$$

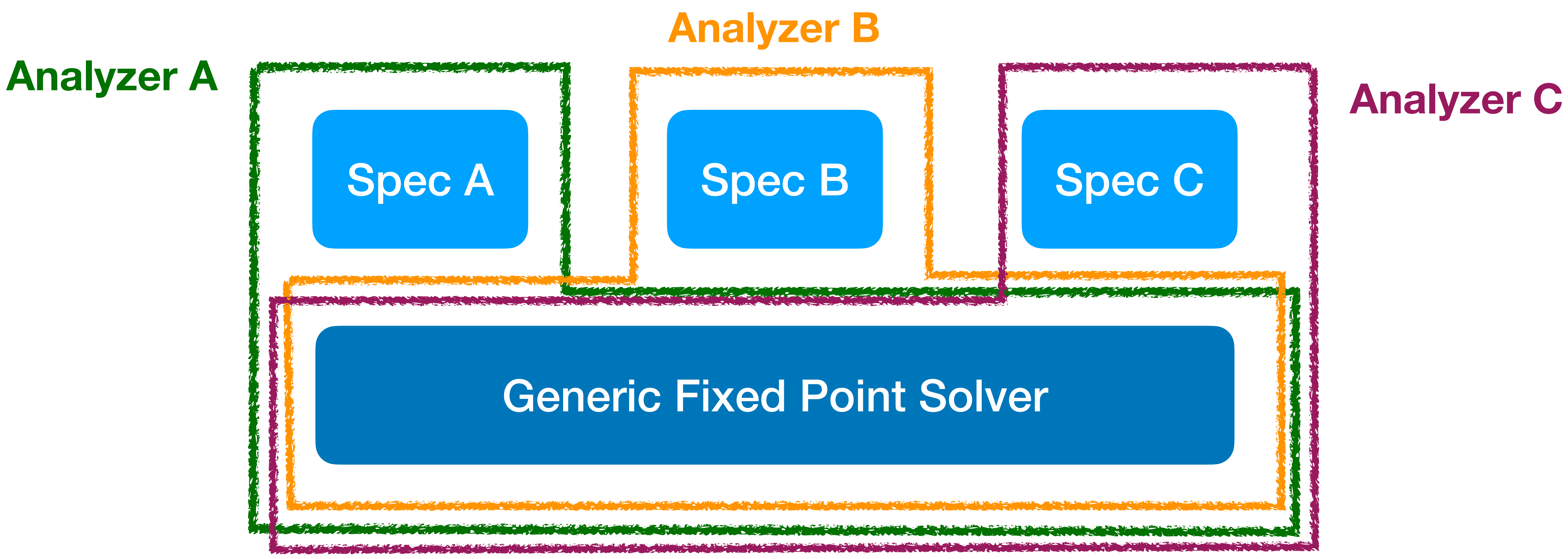
$$\frac{\mathsf{Path}(l_1, l_2) \quad \mathsf{Sink}(l_2)}{\mathsf{Alarm}(l_1, l_2)}$$

Repeat until reaching a fixed point

{ Def$(1,x)$, Src(1), Def$(2,y)$, Use$(2,x)$,
Def$(4,z)$, Use$(4,y)$, San(4),
Sink(6), Use$(6,y)$,
Sink(8), Use$(8,z)$,
Edge$(1,2)$, Edge$(2,4)$, Edge$(2,6)$,
... Alarm$(1,6)$**}**

Implementation

- Static analyzer = constraint specification + constraint solving
  - Specification ("what"): **distinct** rules depending on different purposes
  - Solving ("how"): **generic** fixed point iteration algorithm (boilerplate)



A Constraint Language: Datalog

- A declarative logic programming language
- Not Turing-complete: Datalog = a subset of Prolog = SQL + recursion
  - Always terminate
  - Efficient (polynomial) algorithm to evaluate Datalog programs
- Application: static analysis, DB, network, etc
- Popular solvers: Z3, Souffle, etc

Syntax

- A Datalog program consists of three parts:
  - Input relations: the form of the input to the Datalog program
  - Output relations: the form of the output to the Datalog program
  - Rules: the inference rules that compute the outputs from the inputs

Example: Taint Analysis (1)

- Input relations:

```
Def(l: Location, x: Variable)
Use(l: Location, x: Variable)
Src(l: Location)
San(l: Location)
Sink(l: Location)
```

- Output relations:

```
Edge(l1: Location, l2: Location)
Path(l1: Location, l2: Location)
Alarm(l1: Location, l2: Location)
```

- Rules:

$$\boxed{\frac{\mathsf{Def}(l_1, x) \quad \mathsf{Use}(l_2, x)}{\mathsf{Edge}(l_1, l_2)}}$$

```
Edge(l1, l2)  :- Def(l1, x), Use(l2, x).
Path(l1, l2)  :- Src(l1), Edge(l1, l2).
Path(l1, l3)  :- Path(l1, l2), !San(l2), Edge(l3, l3).
Alarm(l1, l2) :- Path(l1, l2), Sink(l2).
```

Example: Taint Analysis (2)

```
Def(l: Location, x: Variable)
Use(l: Location, x: Variable)
Src(l: Location)
San(l: Location)
Sink(l: Location)
```

```
Edge(l1: Location, l2: Location)
Path(l1: Location, l2: Location)
Alarm(l1: Location, l2: Location)
```

```
Edge(l1, l2)  :- Def(l1, x), Use(l2, x).
Path(l1, l2)  :- Src(l1), Edge(l1, l2).
Path(l1, l3)  :- Path(l1, l2), !San(l2), Edge(l3, l3).
Alarm(l1, l2) :- Path(l1, l2), Sink(l2).
```

Datalog Solver

{Edge(1,2), Edge(2,4), Edge(2,6), Path(1,6), ..., Alarm(1,6)}

Conclusion

- Constraint-based analysis: a specialized framework of static analysis
- Static analysis by monotonic closure
    - Static analysis = setting up initial facts + collecting new facts by rules
- Datalog: a practical domain-specific language for constraint solving
- Limited but powerful enough for simple properties
- Limitations:
    - Sound rules for complicated features?
    - Systematic way to vary the accuracy?