

Program Analysis

15. Static Analysis by Monotonic Closure

Kihong Heo



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:

$$\frac{P_1 \ \cdots \ P_n \text{ premise(s)}}{P \text{ conclusion}} \qquad \frac{human(x) \implies mortal(x) \quad human(s)}{mortal(s)}$$

- Inference rule with zero premise is called an **axiom**
- Examples: inference rules of propositional logic

$$\frac{}{\text{true}} \quad \frac{A \quad B}{A \wedge B} \quad \frac{A \wedge B}{A} \quad \frac{A \wedge B}{B} \quad \frac{A}{A \vee B} \quad \frac{B}{A \vee B} \quad \frac{A \quad A \implies B}{B} \quad \dots$$

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:

$$\begin{array}{ll} C & \rightarrow \begin{array}{l} L := R \\ C; C \\ \text{while } B \ C \end{array} \\ L & \rightarrow x \mid *x \\ R & \rightarrow x \mid *x \mid \&x \\ B & \text{boolean expressions} \end{array}$$

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

All possible points-to facts

$\{x \rightarrow a, y \rightarrow x, x \rightarrow b, \\ a \rightarrow a, b \rightarrow b, a \rightarrow b, b \rightarrow a\}$

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)

$$\frac{x := y \quad y \rightarrow z}{x \rightarrow z}$$

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

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

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

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

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
    
```

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

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

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

Collect the initial facts directly from program text

$$\{x \rightarrow a, y \rightarrow x\}$$

Example

```

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

```

$$\begin{array}{c}
 \frac{x := \&y}{x \rightarrow y} \qquad \frac{x := y \quad y \rightarrow z}{x \rightarrow z} \qquad \frac{*x := y \quad x \rightarrow w \quad y \rightarrow z}{w \rightarrow z} \qquad \boxed{\frac{*x := \&y \quad x \rightarrow w}{w \rightarrow y}} \\
 \\
 \frac{x := *y \quad y \rightarrow z \quad z \rightarrow w}{x \rightarrow w} \qquad \frac{*x := *y \quad x \rightarrow w \quad y \rightarrow z \quad z \rightarrow v}{w \rightarrow v}
 \end{array}$$

Apply the rules to the known facts

$\{x \rightarrow a, y \rightarrow x, x \rightarrow b\}$

Example

```

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

```

$$\begin{array}{c}
 \frac{x := \&y}{x \rightarrow y} \qquad \frac{x := y \quad y \rightarrow z}{x \rightarrow z} \qquad \frac{*x := y \quad x \rightarrow w \quad y \rightarrow z}{w \rightarrow z} \qquad \frac{*x := \&y \quad x \rightarrow w}{w \rightarrow y} \\
 \\
 \frac{x := *y \quad y \rightarrow z \quad z \rightarrow w}{x \rightarrow w} \qquad \boxed{\frac{*x := *y \quad x \rightarrow w \quad y \rightarrow z \quad z \rightarrow v}{w \rightarrow v}}
 \end{array}$$

Repeat until reaching a fixed point

$\{x \rightarrow a, y \rightarrow x, x \rightarrow b, a \rightarrow a, b \rightarrow b, a \rightarrow b, b \rightarrow a\}$

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)

$$\begin{array}{c} \frac{l : x := E}{\text{Def}(l, x)} \quad \frac{l : E := y}{\text{Use}(l, y)} \quad \frac{l : x := \text{source}()}{\text{Src}(l)} \quad \frac{l : x := \text{sanitize}(y)}{\text{San}(l)} \quad \frac{l : x := \text{sanitize}(y)}{\text{Use}(l, y)} \quad \frac{l : \text{sink}(x)}{\text{Sink}(l)} \quad \frac{l : \text{sink}(x)}{\text{Use}(l, x)} \end{array}$$

Inductive cases (deriving new facts)

$$\begin{array}{c} \frac{\text{Def}(l_1, x) \quad \text{Use}(l_2, x)}{\text{Edge}(l_1, l_2)} \quad \frac{\text{Src}(l_1) \quad \text{Edge}(l_1, l_2)}{\text{Path}(l_1, l_2)} \quad \frac{\text{Path}(l_1, l_2) \quad \neg \text{San}(l_2) \quad \text{Edge}(l_2, l_3)}{\text{Path}(l_1, l_3)} \quad \frac{\text{Path}(l_1, l_2) \quad \text{Sink}(l_2)}{\text{Alarm}(l_1, l_2)} \end{array}$$

Example

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

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

Collect the initial facts directly from program text

{ Def(1,x), Src(1), Def(2,y), 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);
```

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

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

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

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) }

Example

```

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

```

$$\begin{array}{c}
 \frac{l : x := E}{\text{Def}(l, x)} \quad \frac{l : E := y}{\text{Use}(l, y)} \quad \frac{l : x := \text{source}()}{\text{Src}(l)} \\
 \\
 \frac{l : x := \text{sanitize}(y)}{\text{San}(l)} \quad \frac{l : x := \text{sanitize}(y)}{\text{Use}(l, y)} \\
 \\
 \boxed{\frac{l : \text{sink}(x)}{\text{Sink}(l)} \quad \frac{l : \text{sink}(x)}{\text{Use}(l, x)}}
 \end{array}$$

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);

```

$$\begin{array}{c}
 \frac{l : x := E}{\text{Def}(l, x)} \quad \frac{l : E := y}{\text{Use}(l, y)} \quad \frac{l : x := \text{source}()}{\text{Src}(l)} \\
 \\
 \frac{l : x := \text{sanitize}(y)}{\text{San}(l)} \quad \frac{l : x := \text{sanitize}(y)}{\text{Use}(l, y)} \\
 \\
 \boxed{\frac{l : \text{sink}(x)}{\text{Sink}(l)} \quad \frac{l : \text{sink}(x)}{\text{Use}(l, x)}}
 \end{array}$$

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

```

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

```

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

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

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);

```

$$\begin{array}{c}
 \frac{\text{Def}(l_1, x) \quad \text{Use}(l_2, x)}{\text{Edge}(l_1, l_2)} \quad \frac{\text{Src}(l_1) \quad \text{Edge}(l_1, l_2)}{\text{Path}(l_1, l_2)} \quad \frac{\text{Path}(l_1, l_2) \quad \neg\text{San}(l_2) \quad \text{Edge}(l_2, l_3)}{\text{Path}(l_1, l_3)} \\
 \\
 \frac{\text{Path}(l_1, l_2) \quad \text{Sink}(l_2)}{\text{Alarm}(l_1, l_2)}
 \end{array}$$

Repeat until reaching a fixed point

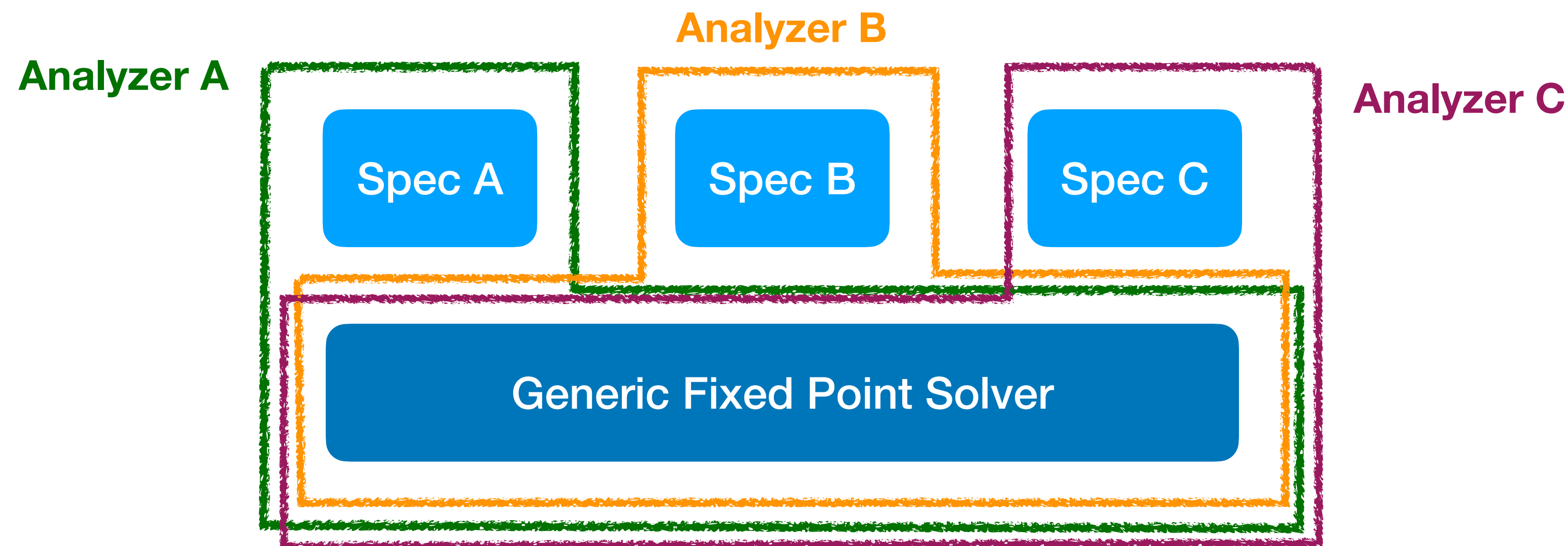
```

{ 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)

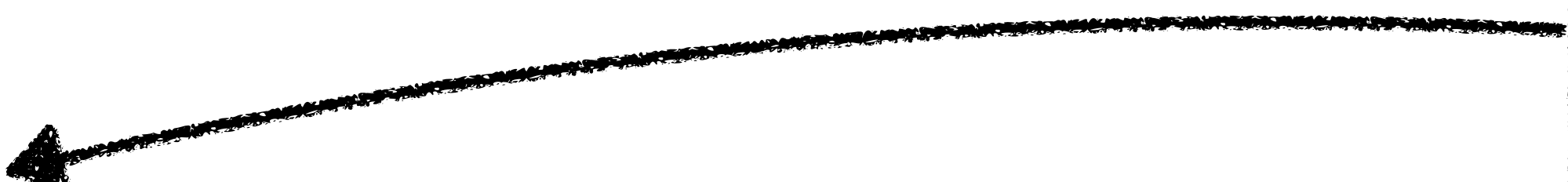
- Output relations:

Edge(l_1 : Location, l_2 : Location)
Path(l_1 : Location, l_2 : Location)
Alarm(l_1 : Location, l_2 : Location)

- Rules:

Edge(l_1, l_2) :- Def(l_1, x), Use(l_2, x).
Path(l_1, l_2) :- Src(l_1), Edge(l_1, l_2).
Path(l_1, l_3) :- Path(l_1, l_2), !San(l_2), Edge(l_3, l_3).
Alarm(l_1, l_2) :- Path(l_1, l_2), Sink(l_2).

Def(l_1, x) Use(l_2, x)
Edge(l_1, l_2)



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?