Lecture 8: Interprocedural Analysis

17-355/17-665/17-819: Program Analysis

Rohan Padhye

September 23, 2025

* Course materials developed with Jonathan Aldrich and Claire Le Goues

Extend WHILE with functions

Extend WHILE3ADDR with functions

$$
\begin{array}{lcl}
F & ::= & \text{fun } f(x)\ \{\ \overline{n : I}\ \} \\
I & ::= & \ldots \mid \text{return } x \mid y := f(x)
\end{array}
$$

Extend WHILE3ADDR with functions

$$
\begin{array}{lcl}
F & ::= & \text{fun } f(x) \; \{ \; \overline{n : I} \; \} \\
I & ::= & \ldots \mid \text{return } x \mid y := f(x)
\end{array}
$$

$$
\begin{array}{ll}
1: & \text{fun } \mathit{double}(x) : \mathit{int} \\
2: & \quad y := 2 * x \\
3: & \quad \text{return } y \\
4: & \text{fun } \mathit{main}() : \mathit{void} \\
5: & \quad z := 0 \\
6: & \quad w := \mathit{double}(z)
\end{array}
$$

Extend WHILE3ADDR with functions

$$
\begin{array}{ll}
1: & \text{fun } \mathit{divByX}(x) : \mathit{int} \\
2: & \quad y := 10/x \\
3: & \quad \text{return } y \\
4: & \text{fun } \mathit{main}() : \mathit{void} \\
5: & \quad z := 5 \\
6: & \quad w := \mathit{divByX}(z)
\end{array}
$$

$$
\begin{array}{ll}
1: & \text{fun } \mathit{double}(x) : \mathit{int} \\
2: & \quad y := 2 * x \\
3: & \quad \text{return } y \\
4: & \text{fun } \mathit{main}() : \mathit{void} \\
5: & \quad z := 0 \\
6: & \quad w := \mathit{double}(z)
\end{array}
$$

How do we analyze these programs?

Data-Flow Analysis

Approach #1: Analyze functions independently

- Pretend function *f()* cannot see the source of function *g()*
- Simulates separate compilation and dynamic linking (e.g. C, Java)
- Create CFG for each function body and run **intraprocedural** analysis
- **Q**: What should σ_0 and $f_Z[\![x := g(y)]\!]$ and $f_Z[\![\text{return } x]\!]$ be for zero analysis?

$$\sigma_0 =$$

$$f[\![x := g(y)]\!](\sigma) =$$

$$f[\![\text{return } x]\!](\sigma) =$$

Can we show that division on line 2 is safe?

$$
\begin{array}{ll}
1: & \text{fun } \mathit{divByX}(x) : \mathit{int} \\
2: & \quad y := 10/x \\
3: & \quad \text{return } y \\
4: & \text{fun } \mathit{main}() : \mathit{void} \\
5: & \quad z := 5 \\
6: & \quad w := \mathit{divByX}(z)
\end{array}
$$

Approach #2: User-defined Annotations

@NonZero -> @NonZero

$$
\begin{array}{l}
\text{fun } \mathit{divByX}(x) : \mathit{int} \\
\quad y := 10/x \\
\quad \text{return } y \\[4pt]
\text{fun } \mathit{main}() : \mathit{void} \\
\quad z := 5 \\
\quad w := \mathit{divByX}(z)
\end{array}
$$

$$
\begin{array}{lll}
f[\![x := g(y)]\!](\sigma) & = \sigma[x \mapsto \mathit{annot}[\![g]\!].r] & (\text{error if } \sigma(y) \not\sqsubseteq \mathit{annot}[\![g]\!].a) \\
f[\![\text{return } x]\!](\sigma) & = \sigma & (\text{error if } \sigma(x) \not\sqsubseteq \mathit{annot}[\![g]\!].r)
\end{array}
$$

Approach #2: User-defined Annotations

@NonZero -> @NonZero

$$
\begin{aligned}
&1: \quad \text{fun } \mathit{divByX}(x) : \mathit{int}\\
&2: \qquad y := 10/x\\
&3: \qquad \text{return } y\\
&4: \quad \text{fun } \mathit{main}() : \mathit{void}\\
&5: \qquad z := 5\\
&6: \qquad w := \mathit{divByX}(z)
\end{aligned}
$$

@NonZero -> @NonZero

$$
\begin{aligned}
&1: \quad \text{fun } \mathit{double}(x) : \mathit{int}\\
&2: \qquad y := 2 * x\\
&3: \qquad \text{return } y\\
&4: \quad \text{fun } \mathit{main}() : \mathit{void}\\
&5: \qquad z := 0\\
&6: \qquad w := \mathit{double}(z)
\end{aligned}
$$
Error!

$$
\begin{aligned}
f[\![x := g(y)]\!](\sigma) &= \sigma[x \mapsto \mathit{annot}[\![g]\!].r] \quad (\text{error if } \sigma(y) \not\sqsubseteq \mathit{annot}[\![g]\!].a)\\
f[\![\text{return } x]\!](\sigma) &= \sigma \quad (\text{error if } \sigma(x) \not\sqsubseteq \mathit{annot}[\![g]\!].r)
\end{aligned}
$$

Approach #2: User-defined Annotations

@NonZero -> @NonZero

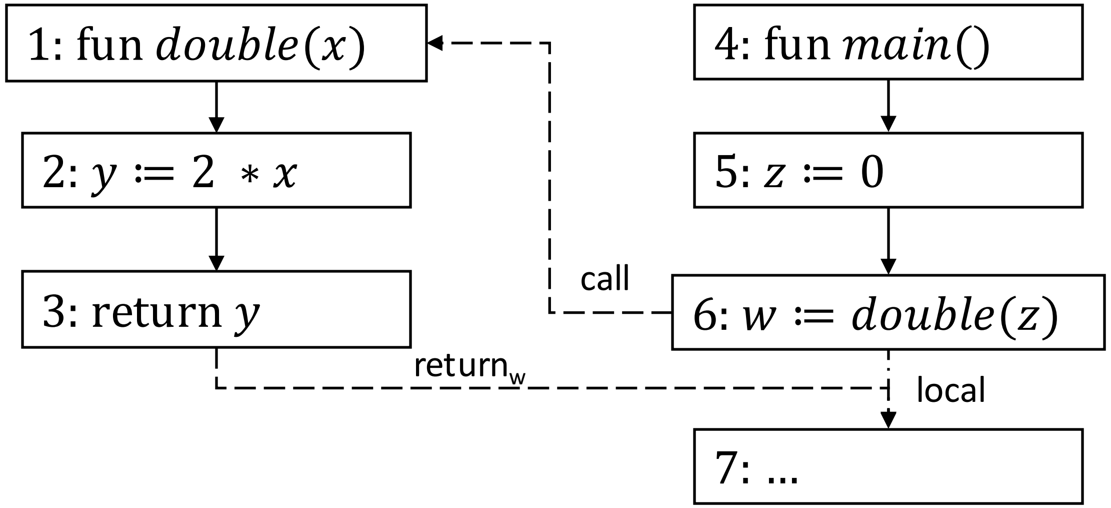
$$
\begin{array}{l}
\text{fun } \mathit{divByX}(x) : \mathit{int} \\
\quad y := 10/x \\
\quad \text{return } y \\
\\
\text{fun } \mathit{main}() : \mathit{void} \\
\quad z := 5 \\
\quad w := \mathit{divByX}(z)
\end{array}
$$

@Any -> @NonZero

$$
\begin{array}{l}
\text{fun } \mathit{double}(x) : \mathit{int} \\
\quad y := 2 * x \\
\quad \text{return } y \quad \textbf{Error!} \\
\\
\text{fun } \mathit{main}() : \mathit{void} \\
\quad z := 0 \\
\quad w := \mathit{double}(z)
\end{array}
$$

$$
\begin{array}{lll}
f[\![x := g(y)]\!](\sigma) & = \sigma[x \mapsto \mathit{annot}[\![g]\!].r] & (\text{error if } \sigma(y) \not\sqsubseteq \mathit{annot}[\![g]\!].a) \\
f[\![\text{return } x]\!](\sigma) & = \sigma & (\text{error if } \sigma(x) \not\sqsubseteq \mathit{annot}[\![g]\!].r)
\end{array}
$$

Approach #3: Interprocedural CFG



$$f_Z[\![x \coloneqq g(y)]\!]_{local}(\sigma) = \sigma \setminus (\{x\} \cup \ Globals)$$

$$f_Z[\![x \coloneqq g(y)]\!]_{call}(\sigma) = \{v \mapsto \sigma(v) |\ v \in Globals\} \cup \{formal(g) \mapsto \sigma(y)\}$$

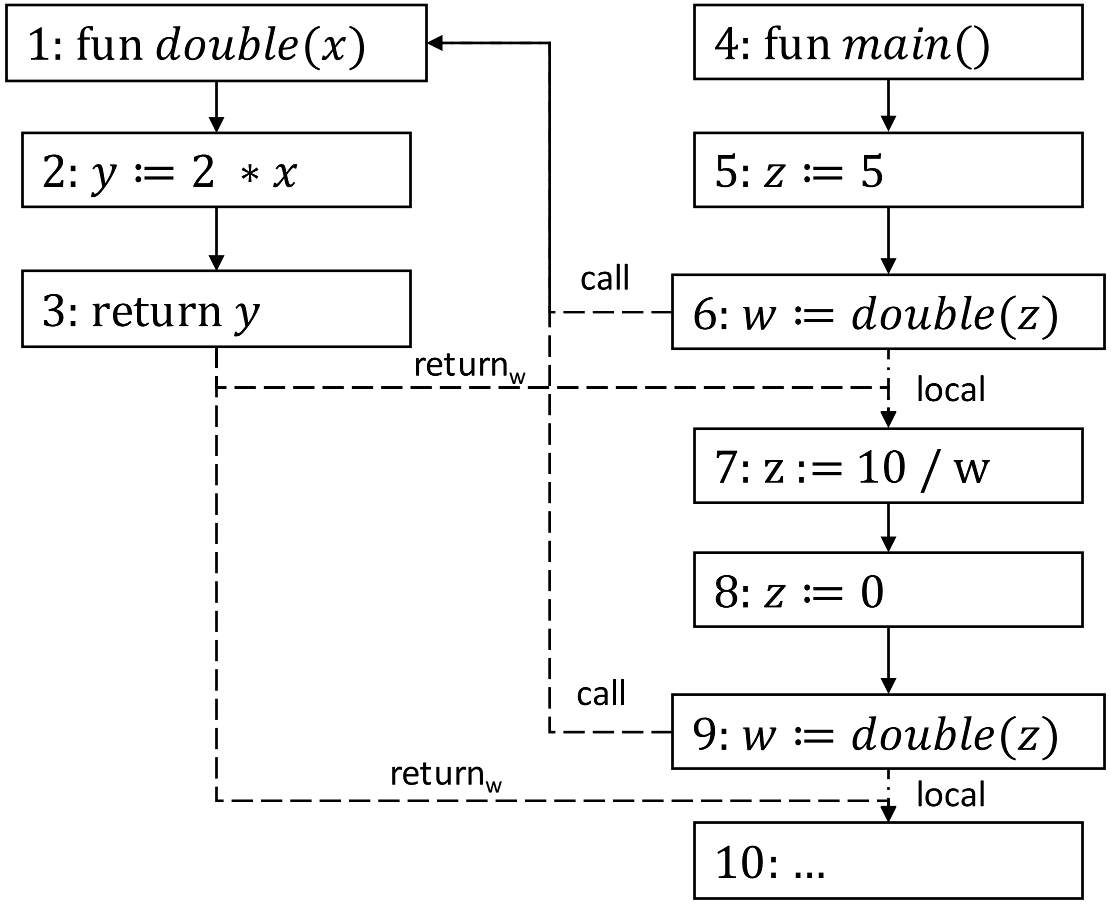
$$f_Z[\![\text{return } y]\!]_{return_x}(\sigma) = \{v \mapsto \sigma(v) |\ v \in Globals\} \cup \{x \mapsto \sigma(y)\}$$

Approach #3: Interprocedural CFG

Exercise: What would be the result of zero analysis for this program at line 7 and at the end (after line 9)?

$$
\begin{array}{ll}
1: & \text{fun } \mathit{double}(x) : \mathit{int} \\
2: & \quad y := 2 * x \\
3: & \quad \text{return } y \\
4: & \text{fun } \mathit{main}() \\
5: & \quad z := 5 \\
6: & \quad w := \mathit{double}(z) \\
7: & \quad z := 10/w \\
8: & \quad z := 0 \\
9: & \quad w := \mathit{double}(z)
\end{array}
$$

Approach #3: Interprocedural CFG



$$
\begin{aligned}
1 &: \quad \text{fun } double(x) : int \\
2 &: \quad\quad y := 2 * x \\
3 &: \quad\quad \text{return } y \\
4 &: \quad \text{fun } main() \\
5 &: \quad\quad z := 5 \\
6 &: \quad\quad w := double(z) \\
7 &: \quad\quad z := 10/w \\
8 &: \quad\quad z := 0 \\
9 &: \quad\quad w := double(z)
\end{aligned}
$$

$$f_Z[\![x \coloneqq g(y)]\!]_{local}(\sigma) = \sigma \setminus (\{x\} \cup \ Globals)$$

$$f_Z[\![x \coloneqq g(y)]\!]_{call}(\sigma) = \{v \mapsto \sigma(v) |\ v \in Globals\} \cup \{formal(g) \mapsto \sigma(y)\}$$

$$f_Z[\![\text{return } y]\!]_{return_x}(\sigma) = \{v \mapsto \sigma(v) |\ v \in Globals\} \cup \{x \mapsto \sigma(y)\}$$

Problems with Interprocedural CFG

- Merges (joins) information across call sites to same function
- Loses precision
- Models infeasible paths (call from one site and return to another)
- Can we “remember” where to return data-flow values?