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?

$$\sigma_0 =$$

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

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

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

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

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

Approach #2: User-defined Annotations

@NonZero -> @NonZero

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

@NonZero -> @NonZero

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

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

Approach #2: User-defined Annotations

@NonZero -> @NonZero

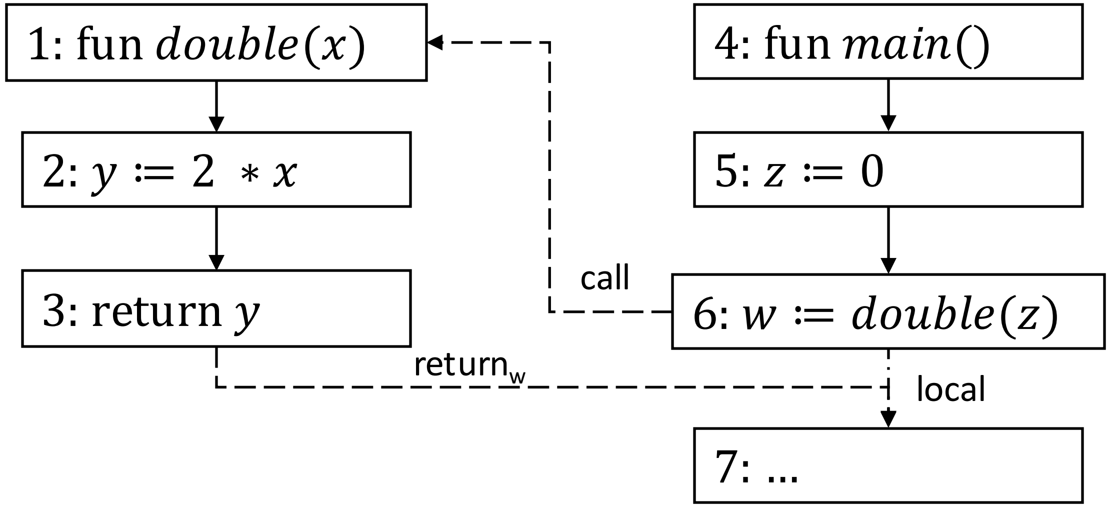
$$
\begin{array}{l}
\text{fun } \mathit{divByX}(x) : \mathit{int} \\
\quad y := 10/x \\
\quad \text{return } y \\
\text{fun } \mathit{main}() : \mathit{void} \\
\quad z := 5 \\
\quad w := \mathit{divByX}(z)
\end{array}
$$

@Any -> @NonZero

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

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

Approach #3: Interprocedural CFG



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

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

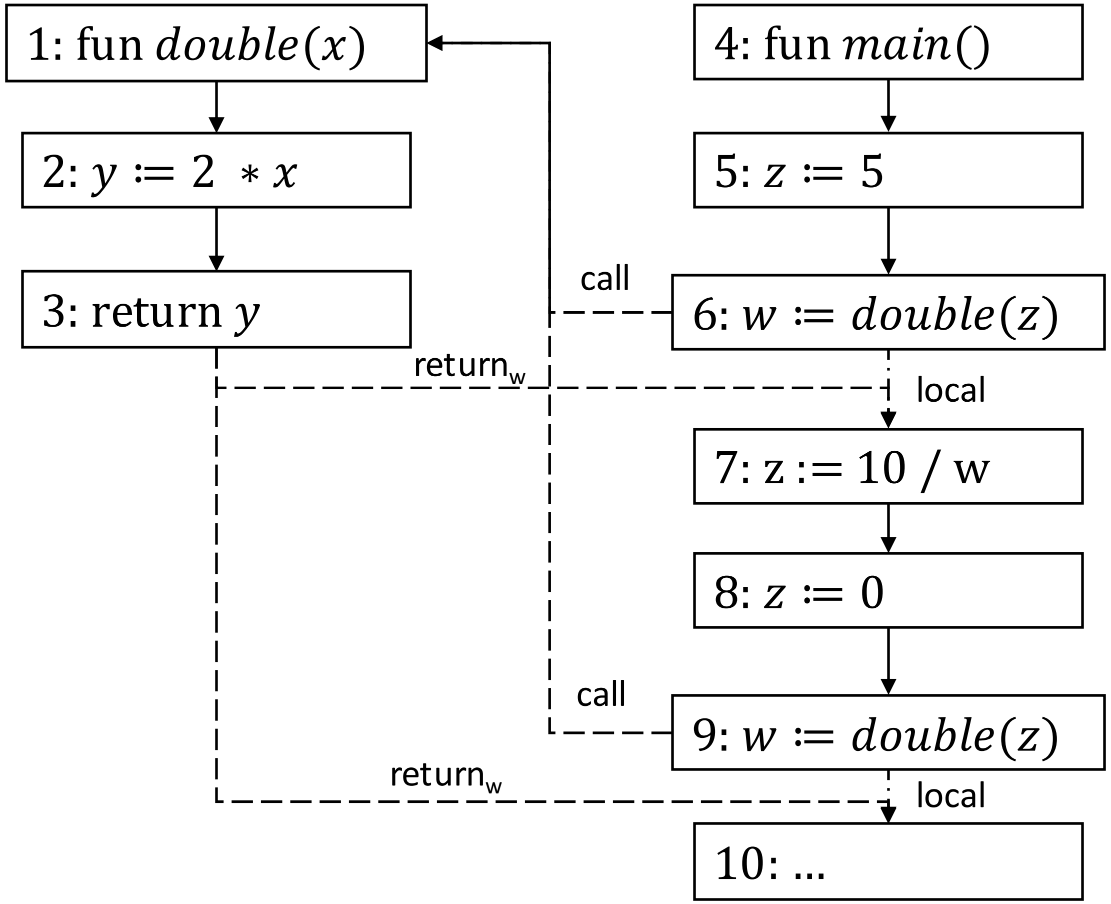
$$f_Z[\![\text{return } y]\!]_{return_x}(\sigma) = \{v \mapsto \sigma(v) |\ v \in Globals\} \cup \{x \mapsto \sigma(y)\}$$

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



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

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

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

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

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?

CONTEXT-SENSITIVE ANALYSIS

Enter:

Context-Sensitive Analysis Example

$$
\begin{array}{l}
\text{fun } \mathit{double}(x) : \mathit{int} \\
\quad y := 2 * x \\
\quad \text{return } y \\
\text{fun } \mathit{main}() \\
\quad z := 5 \\
\quad w := \mathit{double}(z) \\
\quad z := 10/w \\
\quad z := 0 \\
\quad w := \mathit{double}(z)
\end{array}
$$

Key idea: Separate analyses for functions called in different ”contexts”.

(“context” = some statically definable condition)

Context-Sensitive Analysis Example

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

Context	σ_{in}	σ_{out}
Line 6	{x->N}	{x->N, y->N}
Line 9	{x->Z}	{x->Z, y->Z}

Context-Sensitive Analysis Example

```
fun double(x) : int
    y := 2 * x
    return y

fun main()
    z := 5
    w := double(z)
    z := 10/w
    z := 0
    w := double(z)
```

Context	σ_{in}	σ_{out}
<main, T>	T	{w->Z, Z->Z}
<double, N>	{x->N}	{x->N, y->N}
<double, Z>	{x->Z}	{x->Z, y->Z}

type $Context$
val $fn : Function$
val $input : \sigma$

type $Summary$
val $input : \sigma$
val $output : \sigma$

Context	σ_{in}	σ_{out}
<main, T>	T	{w->Z, Z->Z}
<double, N>	{x->N}	{x->N, y->N}
<double, Z>	{x->Z}	{x->Z, y->Z}

Works for non-recursive contexts!

function $\textsc{GetCtx}(f, callingCtx, n, \sigma_{in})$
  return $Context(f, \sigma_{in})$
end function

val $results : Map[Context, Summary]$

function $\textsc{Analyze}(ctx, \sigma_{in})$
  $\sigma'_{out} \leftarrow \textsc{Intraprocedural}(ctx, \sigma_{in})$
  $results[ctx] \leftarrow Summary(\sigma_{in}, \sigma'_{out})$
  return σ'_{out}
end function

function $\textsc{Flow}([\![n\!: x := f(y)]\!], ctx, \sigma_n)$
  $\sigma_{in} \leftarrow [formal(f) \mapsto \sigma_n(y)]$
  $calleeCtx \leftarrow \textsc{GetCtx}(f, ctx, n, \sigma_{in})$
  $\sigma_{out} \leftarrow \textsc{ResultsFor}(calleeCtx, \sigma_{in})$
  return $\sigma_n[x \mapsto \sigma_{out}[result]]$
end function

function $\textsc{ResultsFor}(ctx, \sigma_{in})$
  if $ctx \in \mathrm{dom}(results)$ **then**
    if $\sigma_{in} \sqsubseteq results[ctx].input$ **then**
      return $results[ctx].output$
    else
      return $\textsc{Analyze}(ctx, results[ctx].input \sqcup \sigma_{in})$
    end if
  else
    return $\textsc{Analyze}(ctx, \sigma_{in})$
  end if
end function