Static Program Analysis

Lecture 19: Interprocedural Dataflow Analysis II (Fixpoint Solution)

Winter Semester 2016/17

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-1617/spa/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Elimination of Spurious Counterexamples I

Lemma

If $\langle c_0, \text{true}\rangle \Rightarrow \langle c_1, Q_1\rangle \Rightarrow \ldots \Rightarrow \langle c_k, Q_k\rangle$ *is a spurious counterexample, there exist Boolean expressions* $b_0, \ldots, b_k$ *with* $b_0 \equiv \text{true}$, $b_k \equiv \text{false}$, *and*

$$\forall i \in \{1, \ldots, k\}, \sigma, \sigma' \in \Sigma : \sigma \models b_{i-1} \wedge \langle c_{i-1}, \sigma\rangle \rightarrow \langle c_i, \sigma'\rangle \implies \sigma' \models b_i$$

Proof (idea).

Inductive definition of b_i as strongest postconditions:

1. $b_0 := \text{true}$
2. for $i = 1, \ldots, k$: definition of b_i depending on b_{i-1} and on (axiom) transition rule applied in $\langle c_{i-1}, .\rangle \Rightarrow \langle c_i, .\rangle$:
   - (skip) $b_i := b_{i-1}$
   - (asgn) $b_i := \exists x'.(b_{i-1}[x \mapsto x'] \wedge x = a[x \mapsto x'])$
     (for $x := a$; x' = previous value of x)
   - (if1) $b_i := b_{i-1} \wedge b$
   - (if2) $b_i := b_{i-1} \wedge \neg b$
   - (wh1) $b_i := b_{i-1} \wedge b$
   - (wh2) $b_i := b_{i-1} \wedge \neg b$

(yields $b_k \equiv \text{false}$; by induction on k) □

Recap: The CEGAR Refinement Step

Elimination of Spurious Counterexamples II

Example

- Let $c_0 := [\mathtt{x := z}]^0; [\mathtt{z := z + 1}]^1; [\mathtt{y := z}]^2;$
  $\mathtt{if}\ [\mathtt{x = y}]^3\ \mathtt{then}\ [\mathtt{skip}]^4\ \mathtt{else}\ [\mathtt{skip}]^5\ \mathtt{end}$
- Interesting property: after termination, $\mathtt{x} \neq \mathtt{y}$, i.e., label 4 unreachable
- Initial abstraction: $P = \emptyset$ ($\Longrightarrow$ $Abs(P) = \{\text{true}, \text{false}\}$)
- (Spurious) counterexample: $\langle 0, \text{true}\rangle \Rightarrow \langle 1, \text{true}\rangle \Rightarrow \langle 2, \text{true}\rangle \Rightarrow \langle 3, \text{true}\rangle \Rightarrow \langle 4, \text{true}\rangle$
- Forward construction of strongest postconditions:
  - $b_0 := \text{true}$
  - (asgn) $b_i := \exists x'.(b_{i-1}[x \mapsto x'] \wedge x = a[x \mapsto x'])$
    $\Longrightarrow$ $b_1 := \exists x'.(b_0[\mathtt{x} \mapsto x'] \wedge \mathtt{x} = \mathtt{z}[\mathtt{x} \mapsto x']) \equiv (\mathtt{x} = \mathtt{z})$
  - (asgn) $b_i := \exists x'.(b_{i-1}[x \mapsto x'] \wedge x = a[x \mapsto x'])$
    $\Longrightarrow$ $b_2 := \exists z'.(b_1[\mathtt{z} \mapsto z'] \wedge \mathtt{z} = \mathtt{z} + 1[\mathtt{z} \mapsto z'])$
    $= \exists z'.(\mathtt{x} = z' \wedge \mathtt{z} = z' + 1) \equiv (\mathtt{x} + 1 = \mathtt{z})$
  - (asgn) $b_i := \exists x'.(b_{i-1}[x \mapsto x'] \wedge x = a[x \mapsto x'])$
    $\Longrightarrow$ $b_3 := \exists y'.(b_2[\mathtt{y} \mapsto y'] \wedge \mathtt{y} = \mathtt{z}[\mathtt{y} \mapsto y']) \equiv (\mathtt{x} + 1 = \mathtt{z} \wedge \mathtt{y} = \mathtt{z})$
  - (if1) $b_i := b_{i-1} \wedge b$
    $\Longrightarrow$ $b_4 := (b_3 \wedge \mathtt{x} = \mathtt{y}) \equiv (\mathtt{x} + 1 = \mathtt{z} \wedge \mathtt{y} = \mathtt{z} \wedge \mathtt{x} = \mathtt{y}) \equiv \text{false}$

Recap: The CEGAR Refinement Step

Abstraction Refinement

- Using $b_1, \ldots, b_{k-1}$ as computed before, let $P' := P \cup \{p_1, \ldots, p_n\}$ where $p_1, \ldots, p_n$ are the atomic conjuncts occurring in $b_1, \ldots, b_{k-1}$
- Refine $Abs(P)$ to $Abs(P')$

Lemma

After refinement, the spurious counterexample

$$\langle c_0, \text{true} \rangle \Rightarrow \langle c_1, Q_1 \rangle \Rightarrow \ldots \Rightarrow \langle c_k, Q_k \rangle$$

with $Q_k \not\equiv \text{false}$ *does not exist anymore.*

Proof.

omitted □

Abstract Semantics for Predicate Abstraction

Definition (Execution relation for predicate abstraction)

If $c \in \mathit{Cmd}$ and $Q \in \mathit{Abs}(P)$, then $\langle c, Q \rangle$ is called an abstract configuration. The execution relation for predicate abstraction is defined by the following rules:

$$\text{(skip)}\ \frac{}{\langle \texttt{skip}, Q \rangle \Rightarrow \langle \downarrow, Q \rangle} \qquad \text{(asgn)}\ \frac{}{\langle x := a, Q \rangle \Rightarrow \langle \downarrow, \bigsqcup \{ Q_{\sigma[x \mapsto \mathit{val}_\sigma(a)]} \mid \sigma \models Q \} \rangle}$$

$$\text{(seq1)}\ \frac{\langle c_1, Q \rangle \Rightarrow \langle c_1', Q' \rangle \quad c_1' \neq \downarrow}{\langle c_1 ; c_2, Q \rangle \Rightarrow \langle c_1' ; c_2, Q' \rangle} \qquad \text{(seq2)}\ \frac{\langle c_1, Q \rangle \Rightarrow \langle \downarrow, Q' \rangle}{\langle c_1 ; c_2, Q \rangle \Rightarrow \langle c_2, Q' \rangle}$$

$$\text{(if1)}\ \frac{}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}, Q \rangle \Rightarrow \langle c_1, \overline{Q \wedge b} \rangle}$$

$$\text{(if2)}\ \frac{}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}, Q \rangle \Rightarrow \langle c_2, \overline{Q \wedge \neg b} \rangle}$$

$$\text{(wh1)}\ \frac{}{\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}, Q \rangle \Rightarrow \langle c ; \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}, \overline{Q \wedge b} \rangle}$$

$$\text{(wh2)}\ \frac{}{\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}, Q \rangle \Rightarrow \langle \downarrow, \overline{Q \wedge \neg b} \rangle}$$

A Simple Example

Example

- Let $c_0 := [\texttt{x := z}]^0; [\texttt{z := z + 1}]^1; [\texttt{y := z}]^2;$
  $\texttt{if } [\texttt{x = y}]^3 \texttt{ then } [\texttt{skip}]^4 \texttt{ else } [\texttt{skip}]^5 \texttt{ end}$
- $P = \emptyset$, $P' = \{\underbrace{\texttt{x} = \texttt{z}}_{p_1}, \underbrace{\texttt{x} + 1 = \texttt{z}}_{p_2}, \underbrace{\texttt{y} = \texttt{z}}_{p_3}\}$
- Refined abstract transitions:

$$\begin{aligned}\langle 0, \text{true}\rangle &\Rightarrow \langle 1, p_1 \wedge \neg p_2\rangle \\ &\Rightarrow \langle 2, \neg p_1 \wedge p_2\rangle \\ &\Rightarrow \langle 3, \neg p_1 \wedge p_2 \wedge p_3\rangle \\ &\Rightarrow \langle 4, \underbrace{\overline{\neg p_1 \wedge p_2 \wedge p_3 \wedge \texttt{x=y}}}_{\equiv \text{ false}}\rangle\end{aligned}$$

Another Example: Multiplication

Example

- Let $c_0 :=$
```
[z := 0]^0;
while [x > 0]^1 do
  [z := z + y]^2;
  [x := x - 1]^3
end;
if [z mod y = 0]^4 then
  [skip]^5
else
  [skip]^6
end;
```
$p_3 = (\texttt{z mod y} \neq \texttt{0})$ not required!
- Global assumption: $\texttt{y} > 0$
- Interesting property: label 6 unreachable (since $\texttt{z}$ multiple of $\texttt{y}$)
- Initial abstraction: $P = \emptyset$ ($\Longrightarrow$ $Abs(P) = \{\text{true}, \text{false}\}$)
- Abstraction refinement: on the board

Recap: Interprocedural Dataflow Analysis

Extending the Syntax

Syntactic categories:

Category	Domain	Meta variable
Procedure identifiers	$\mathit{Pid} = \{\mathtt{P}, \mathtt{Q}, \ldots\}$	P
Procedure declarations	PDec	p
Commands (statements)	Cmd	c

Context-free grammar:

$$p ::= \mathtt{proc}\ [P(\mathtt{val}\ x, \mathtt{res}\ y)]^{l_n}\ \mathtt{is}\ c\ [\mathtt{end}]^{l_x}; p \mid \varepsilon \in \mathit{PDec}$$
$$c ::= [\mathtt{skip}]^{l} \mid [x := a]^{l} \mid c_1; c_2 \mid \mathtt{if}\ [b]^{l}\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ \mathtt{end} \mid$$
$$\mathtt{while}\ [b]^{l}\ \mathtt{do}\ c\ \mathtt{end} \mid [\mathtt{call}\ P(a, x)]^{l_c}_{l_r} \in \mathit{Cmd}$$

- All labels and procedure names in program $p\,c$ distinct
- In $\mathtt{proc}\ [P(\mathtt{val}\ x, \mathtt{res}\ y)]^{l_n}\ \mathtt{is}\ c\ [\mathtt{end}]^{l_x}$, l_n / l_x refers to the entry / exit of P
- In $[\mathtt{call}\ P(a, x)]^{l_c}_{l_r}$, l_c/l_r refers to the call of / return from P
- First parameter call-by-value (input), second call-by-result (output)

Recap: Interprocedural Dataflow Analysis

Naive Formulation I

- **Attempt:** directly transfer techniques from intraprocedural analysis
  $\Longrightarrow$ treat $(l_c; l_n)$ like (l_c, l_n) and $(l_x; l_r)$ like (l_x, l_r)
- Given: dataflow system $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$
- For each procedure call $[\texttt{call } P(a,x)]^{l_c}_{l_r}$:
  transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \to D$ (definition later)
- For each procedure declaration $\texttt{proc } [P(\texttt{val } x, \texttt{res } y)]^{l_n} \texttt{ is } c\ [\texttt{end}]^{l_x}$:
  transfer functions $\varphi_{l_n}, \varphi_{l_x} : D \to D$ (definition later)
- Induces equation system

$$\mathsf{AI}_l = \begin{cases} \iota & \text{if } l \in E \\ \bigsqcup\{\varphi_{l'}(\mathsf{AI}_{l'}) \mid (l', l) \in F \text{ or } (l'; l) \in F\} & \text{otherwise} \end{cases}$$

- **Problem:** procedure calls $(l_c; l_n)$ and procedure returns $(l_x; l_r)$ treated like goto's
  $\Longrightarrow$ nesting of calls and returns ignored
  $\Longrightarrow$ too many paths considered
  $\Longrightarrow$ analysis information possibly imprecise (but still correct)

Naive Formulation II

Example (Impreciseness of constant propagation analysis)

```
proc [P(val x, res y)]^1 is
  [y := x]^2
[end]^3;
if [y = 0]^4 then
  [call P(1, y)]^5_6;
  [y := y - 1]^7
else
  [call P(2, y)]^8_9;
  [y := y - 2]^10
end;
[skip]^11
```

Two "valid" and two "invalid" paths:

- Valid: $[4, 5, 1, 2, 3, 6, 7, 11]$
  $\Longrightarrow$ $y = 0$ at label 11
- Valid: $[4, 8, 1, 2, 3, 9, 10, 11]$
  $\Longrightarrow$ $y = 0$ at label 11
- Invalid: $[4, 5, 1, 2, 3, 9, 10, 11]$
  $\Longrightarrow$ $y = -1$ at label 11
- Invalid: $[4, 8, 1, 2, 3, 6, 7, 11]$
  $\Longrightarrow$ $y = 1$ at label 11

$\Longrightarrow$ actually always $y = 0$ at 11, but naive method yields $y = \top$

The MVP Solution I

Definition (Complete valid paths)

Let $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system. For every $l \in Lab$, the set of valid paths up to l is given by

$$VPath(l) := \{[l_1, \ldots, l_{k-1}] \mid k \geq 1, l_1 \in E, l_k = l, [l_1, \ldots, l_k] \text{ valid path from } l_1 \text{ to } l_k\}.$$

For $\pi = [l_1, \ldots, l_{k-1}] \in VPath(l)$, we define the transfer function $\varphi_\pi : D \to D$ by

$$\varphi_\pi := \varphi_{l_{k-1}} \circ \ldots \circ \varphi_{l_1} \circ \mathrm{id}_D$$

(so that $\varphi_{[]} = \mathrm{id}_D$).

The MVP Solution II

Definition (MVP solution)

Let $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system where $Lab = \{l_1, \ldots, l_n\}$.
The MVP solution for S is determined by

$$\mathrm{mvp}(S) := (\mathrm{mvp}(l_1), \ldots, \mathrm{mvp}(l_n)) \in D^n$$

where, for every $l \in Lab$,

$$\mathrm{mvp}(l) := \bigsqcup \{\varphi_\pi(\iota) \mid \pi \in VPath(l)\}.$$

Corollary

1. $\mathrm{mvp}(S) \sqsubseteq \mathrm{mop}(S)$
2. *The MVP solution is undecidable.*

Proof.

1. since $VPath(l) \subseteq Path(l)$ for every $l \in Lab$
2. as $\mathrm{mvp}(S) = \mathrm{mop}(S)$ in intraprocedural case and MOP solution undecidable (Thm. 7.1) □

Making Context Explicit

- **Goal:** adapt fixpoint solution to avoid invalid paths
- **Approach:** encode call history into data flow properties
  (use stacks D^+ as dataflow version of runtime stack)
- Non-procedural constructs (`skip`, assignments, tests): operate only on topmost element
- call: computes new topmost entry from current and pushes it
- return: removes topmost entry and combines it with underlying (= call-site) entry

The Interprocedural Extension I

Definition 19.1 (Interprocedural extension (forward analysis))

Let $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system where $\varphi_{l_r} : D^2 \to D$ for each $(l_c, l_n, l_x, l_r) \in \text{iflow}$ (and $\varphi_l : D \to D$ otherwise).
The interprocedural extension of S is given by

$$\hat{S} := (Lab, E, F, (\hat{D}, \hat{\sqsubseteq}), \hat{\iota}, \hat{\varphi})$$

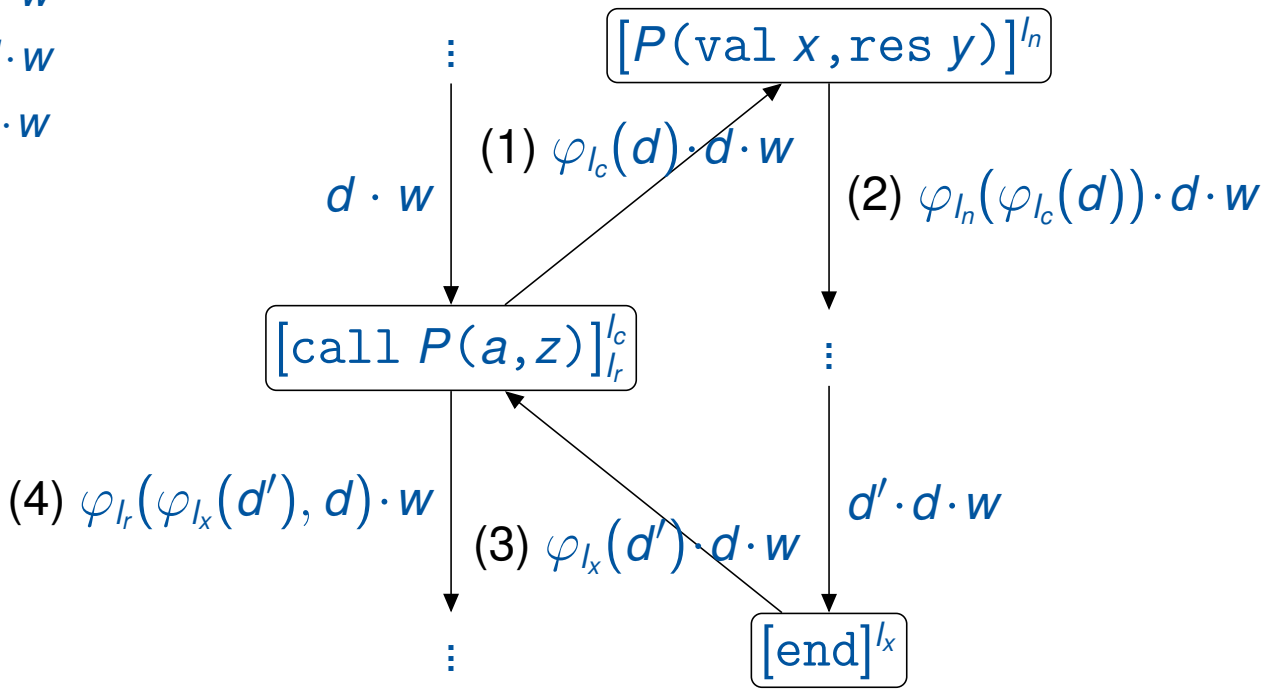
where

- $\hat{D} := D^+$
- $d_1 \dots d_n \hat{\sqsubseteq} d'_1 \dots d'_n$ iff $d_i \sqsubseteq d'_i$ for every $1 \leq i \leq n$
- $\hat{\iota} := \iota \in D^+$
- $\hat{\varphi}_l : D^+ \to D^+$ where
  - for each $l \in Lab \setminus \{l_c, l_r \mid (l_c, l_n, l_x, l_r) \in \text{iflow}\}$:
    $$\hat{\varphi}_l(d \cdot w) := \varphi_l(d) \cdot w$$
  - for each $(l_c, l_n, l_x, l_r) \in \text{iflow}$:
    $$\hat{\varphi}_{l_c}(d \cdot w) := \varphi_{l_c}(d) \cdot d \cdot w$$
    $$\hat{\varphi}_{l_r}(d' \cdot d \cdot w) := \varphi_{l_r}(d', d) \cdot w$$

The Interprocedural Extension II

Flow of information:

1. $\hat{\varphi}_{l_c}(d \cdot w) = \varphi_{l_c}(d) \cdot d \cdot w$
2. $\hat{\varphi}_{l_n}(d' \cdot d \cdot w) = \varphi_{l_n}(d') \cdot d \cdot w$
3. $\hat{\varphi}_{l_x}(d' \cdot d \cdot w) = \varphi_{l_x}(d') \cdot d \cdot w$
4. $\hat{\varphi}_{l_r}(d' \cdot d \cdot w) = \varphi_{l_r}(d', d) \cdot w$



The Interprocedural Extension III

Example 19.2 (Constant Propagation; cf. Lecture 5/6)

$\hat{S} := (Lab, E, F, (\hat{D}, \hat{\sqsubseteq}), \hat{\iota}, \hat{\varphi})$ is determined by

- $D := \{\delta \mid \delta : Var_c \to \mathbb{Z} \cup \{\bot, \top\}\}$ (constant/undefined/overdefined)
- $\bot \sqsubseteq z \sqsubseteq \top$ for every $z \in \mathbb{Z}$
- $\iota := \delta_\top \in D$
- For each $l \in Lab \setminus \{l_c, l_n, l_x, l_r \mid (l_c, l_n, l_x, l_r) \in \text{iflow}\}$,

$$\varphi_l(\delta) := \begin{cases} \delta & \text{if } B^l = \texttt{skip} \text{ or } B^l \in BExp \\ \delta[x \mapsto val_\delta(a)] & \text{if } B^l = (x := a) \end{cases}$$

- Whenever $p\,c$ contains $[\texttt{call } P(a,z)]^{l_c}_{l_r}$ and $\texttt{proc } [P(\texttt{val } x, \texttt{res } y)]^{l_n} \texttt{ is } c\ [\texttt{end}]^{l_x}$,
  - call/entry: set input and reset output parameter

$$\varphi_{l_c}(\delta) := \delta[x \mapsto val_\delta(a), y \mapsto \top], \quad \varphi_{l_n}(\delta) := \delta$$

  - exit/return: reset parameters (x, y possible used in calling context) and set return value

$$\varphi_{l_x}(\delta) := \delta, \quad \varphi_{l_r}(\delta', \delta) := \delta'[x \mapsto \delta(x), y \mapsto \delta(y), z \mapsto \delta'(y)]$$

Types of Equations

For an interprocedural dataflow system $\hat{S} := (Lab, E, F, (\hat{D}, \hat{\sqsubseteq}), \hat{\iota}, \hat{\varphi})$, the intraprocedural equation system (cf. Definition 4.9)

$$\mathsf{AI}_l = \begin{cases} \iota & \text{if } l \in E \\ \bigsqcup\{\varphi_{l'}(\mathsf{AI}_{l'}) \mid (l', l) \in F\} & \text{otherwise} \end{cases}$$

is extended to a system with three kinds of equations (for every $l \in Lab$):

- for actual dataflow information: $\mathsf{AI}_l \in \hat{D}$
  - counterpart of intraprocedural AI
- for transfer functions of single nodes: $f_l : \hat{D} \to \hat{D}$
  - extension of intraprocedural transfer functions by special handling of procedure calls
- for transfer functions of complete procedures: $F_l : \hat{D} \to \hat{D}$
  - $F_l(w)$ yields information at l if corresponding procedure is called with information w
  - thus complete procedure represented by F_{l_x} (“procedure summary”)

Formal Definition of Equation System

Dataflow equations:

$$\mathsf{AI}_l = \begin{cases} \iota & \text{if } l \in E \\ \mathsf{AI}_{l_c} & \text{if } l = l_r \text{ for some } (l_c, l_n, l_x, l_r) \in \text{iflow} \\ \bigsqcup \{ f_{l'}(\mathsf{AI}_{l'}) \mid (l', l) \in F \} & \text{otherwise} \end{cases}$$

Node transfer functions (if l not an exit label):

$$f_l(w) = \begin{cases} \hat{\varphi}_{l_r}(\hat{\varphi}_{l_x}(F_{l_x}(\hat{\varphi}_{l_c}(w)))) & \text{if } l = l_r \text{ for some } (l_c, l_n, l_x, l_r) \in \text{iflow} \\ \hat{\varphi}_l(w) & \text{otherwise} \end{cases}$$

Procedure transfer functions (if l occurs in some procedure):

$$F_l(w) = \begin{cases} w & \text{if } l = l_n \text{ for some } (l_c, l_n, l_x, l_r) \in \text{iflow} \\ \bigsqcup \{ f_{l'}(F_{l'}(w)) \mid (l', l) \in F \} & \text{otherwise} \end{cases}$$

As before: induces monotonic functional on lattice with ACC

$\Longrightarrow$ least fixpoint effectively computable

Example of Equation System

Example 19.3 (Constant Propagation)

Program:

```
proc [P(val x, res y)]^1 is
  [y := 2*(x-1)]^2;
[end]^3;
[call P(2, z)]^4_5;
[call P(z, z)]^6_7;
[skip]^8
```

Dataflow equations:

$$\begin{aligned}
\mathsf{AI}_1 &= f_4(\mathsf{AI}_4) \sqcup f_6(\mathsf{AI}_6)\\
\mathsf{AI}_2 &= f_1(\mathsf{AI}_1)\\
\mathsf{AI}_3 &= f_2(\mathsf{AI}_2)\\
\mathsf{AI}_4 &= \iota = \top\top\top\\
\mathsf{AI}_5 &= \mathsf{AI}_4\\
\mathsf{AI}_6 &= f_5(\mathsf{AI}_5)\\
\mathsf{AI}_7 &= \mathsf{AI}_6\\
\mathsf{AI}_8 &= f_7(\mathsf{AI}_7)
\end{aligned}$$

Fixpoint iteration:

on the board

Node transfer functions:

$$\begin{aligned}
\hat{\varphi}_1(\delta w) &= \delta w\\
\hat{\varphi}_2(\delta w) &= \delta[\mathtt{y} \mapsto \mathit{val}_\delta(\mathtt{2*(x-1)})]w\\
\hat{\varphi}_3(\delta w) &= \delta w\\
\hat{\varphi}_4(\delta w) &= \delta[\mathtt{x} \mapsto 2, \mathtt{y} \mapsto \top]\delta w\\
\hat{\varphi}_5(\delta'\delta w) &= \delta'[\mathtt{x} \mapsto \delta(\mathtt{x}), \mathtt{y} \mapsto \delta(\mathtt{y}), \mathtt{z} \mapsto \delta'(\mathtt{y})]w\\
\hat{\varphi}_6(\delta w) &= \delta[\mathtt{x} \mapsto \delta(\mathtt{z}), \mathtt{y} \mapsto \top]\delta w\\
\hat{\varphi}_7(\delta'\delta w) &= \delta'[\mathtt{x} \mapsto \delta(\mathtt{x}), \mathtt{y} \mapsto \delta(\mathtt{y}), \mathtt{z} \mapsto \delta'(\mathtt{y})]w
\end{aligned}$$

$$\begin{aligned}
f_1(\delta w) &= \hat{\varphi}_1(\delta w) = \delta w\\
f_2(\delta w) &= \hat{\varphi}_2(\delta w) = \delta[\mathtt{y} \mapsto \mathit{val}_\delta(\mathtt{2*(x-1)})]w\\
f_4(\delta w) &= \hat{\varphi}_4(\delta w) = \delta[\mathtt{x} \mapsto 2, \mathtt{y} \mapsto \top]\delta w\\
f_5(\delta w) &= \hat{\varphi}_5(\hat{\varphi}_3(F_3(\hat{\varphi}_4(\delta w)))) = \hat{\varphi}_5(F_3(\hat{\varphi}_4(\delta w)))\\
f_6(\delta w) &= \hat{\varphi}_6(\delta w) = \delta[\mathtt{x} \mapsto \delta(\mathtt{z}), \mathtt{y} \mapsto \top]\delta w\\
f_7(\delta w) &= \hat{\varphi}_7(\hat{\varphi}_3(F_3(\hat{\varphi}_6(\delta w)))) = \hat{\varphi}_7(F_3(\hat{\varphi}_6(\delta w)))\\
f_8(\delta w) &= \hat{\varphi}_8(\delta w) = \delta w
\end{aligned}$$

Procedure transfer functions:

$$\begin{aligned}
F_1(\delta w) &= \delta w\\
F_2(\delta w) &= f_1(F_1(\delta w)) = \delta w\\
F_3(\delta w) &= f_2(F_2(\delta w)) = \delta[\mathtt{y} \mapsto \mathit{val}_\delta(\mathtt{2*(x-1)})]w
\end{aligned}$$