



Static Program Analysis

Lecture 17: Interprocedural Dataflow Analysis I (MVP Solution)

Summer Semester 2018

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ss-18/spa/>

Interprocedural Dataflow Analysis

Outline of Lecture 17

Interprocedural Dataflow Analysis

Procedure Flow Graphs

Intraprocedural vs. Interprocedural Analysis

The MVP Solution

Interprocedural Dataflow Analysis

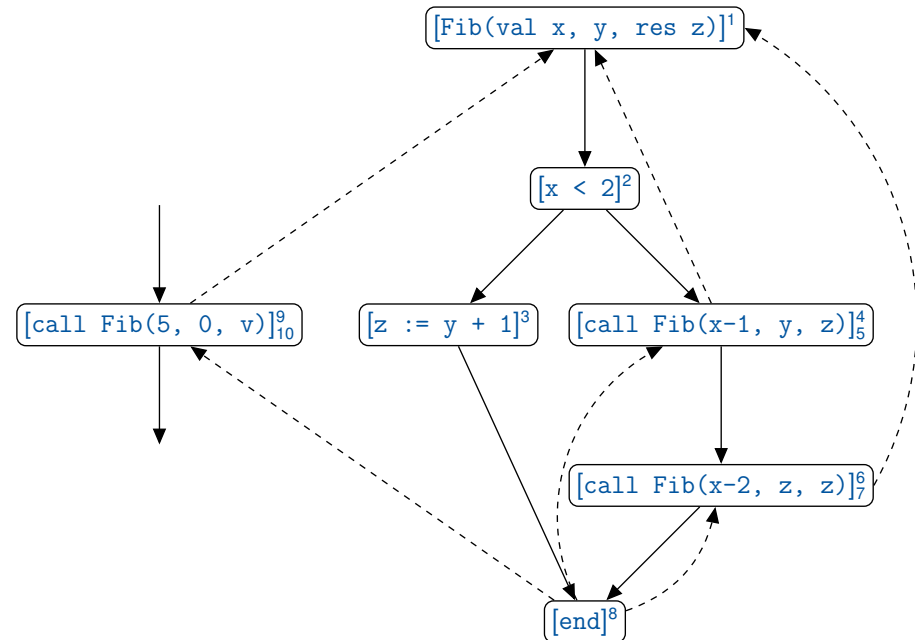
Overview

- **So far:** only **intraprocedural analyses** (i.e., without user-defined functions or procedures or just within their bodies)

Interprocedural Dataflow Analysis

Overview

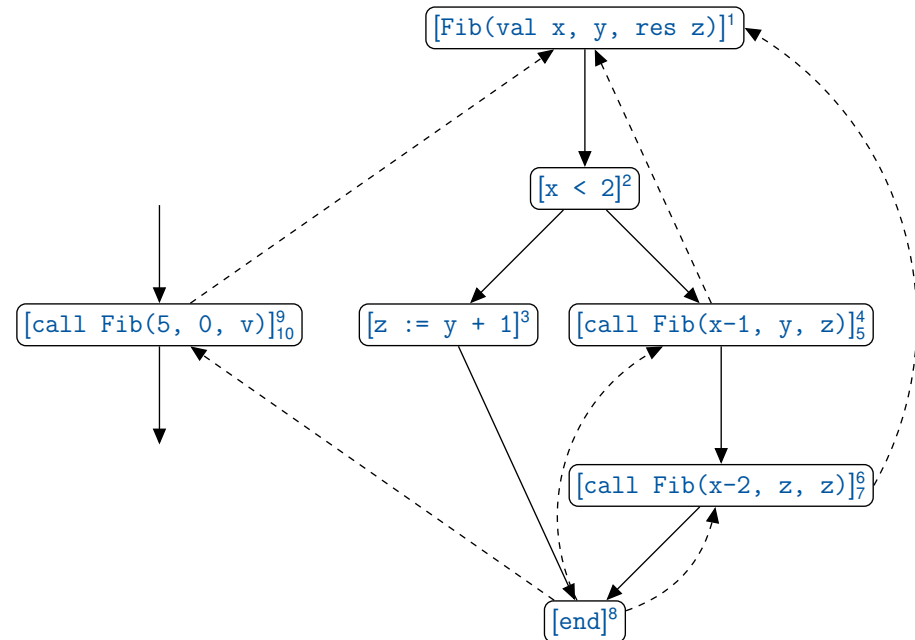
- **So far:** only **intraprocedural analyses** (i.e., without user-defined functions or procedures or just within their bodies)
- **Now:** **interprocedural dataflow analysis** to incorporate effects of
 - calling contexts in (body of) called procedure
 - procedure calls in calling context



Interprocedural Dataflow Analysis

Overview

- **So far:** only **intraprocedural analyses** (i.e., without user-defined functions or procedures or just within their bodies)
- **Now:** **interprocedural dataflow analysis** to incorporate effects of
 - calling contexts in (body of) called procedure
 - procedure calls in calling context
- **Complications:**
 - correct **matching** between calls and returns
 - **parameter passing** (aliasing effects)



Our Approach

- Generic approaches:
 - **functional** approach (here)
 - introduces procedure summaries
 - represent combined effect of procedure execution
 - **call string** approach
 - abstraction of stack-based operational semantics
 - controls level of preciseness (and complexity of analysis) by stack depth
(depth = 0 \implies call $\hat{=}$ goto)

Our Approach

- Generic approaches:
 - **functional** approach (here)
 - introduces procedure summaries
 - represent combined effect of procedure execution
 - **call string** approach
 - abstraction of stack-based operational semantics
 - controls level of preciseness (and complexity of analysis) by stack depth
(depth = 0 \implies call $\hat{=}$ goto)
- Problem-specific approaches:
 - Constant Propagation
 - Alias/Points-To Analysis
 - Partial Redundancy Elimination
 - ...

Our Approach

- Generic approaches:
 - **functional** approach (here)
 - introduces procedure summaries
 - represent combined effect of procedure execution
 - **call string** approach
 - abstraction of stack-based operational semantics
 - controls level of preciseness (and complexity of analysis) by stack depth
(depth = 0 \implies call $\hat{=}$ goto)
- Problem-specific approaches:
 - Constant Propagation
 - Alias/Points-To Analysis
 - Partial Redundancy Elimination
 - ...
- Here: simple setting:
 - only **top-level declarations**, no blocks or nested declarations
 - mutual **recursion**
 - one call-by-value and one call-by-result **parameter**
(extension to multiple and call-by-value-result parameters straightforward)

Interprocedural Dataflow Analysis

Extending the Syntax

Syntactic categories:

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

Extending the Syntax

Syntactic categories:

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

Context-free grammar

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

Interprocedural Dataflow Analysis

Extending the Syntax

Syntactic categories:

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

Context-free grammar

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

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

Interprocedural Dataflow Analysis

An Example

Example 17.1 (Fibonacci numbers)

(with extension by multiple call-by-value parameters)

```
proc [Fib(val x, y, res z)]1 is
  if [x < 2]2 then
    [z := y + 1]3
  else
    [call Fib(x-1, y, z)]45;
    [call Fib(x-2, z, z)]67
  end
[end]8;
[call Fib(5, 0, v)]910
```

Procedure Flow Graphs

Outline of Lecture 17

Interprocedural Dataflow Analysis

Procedure Flow Graphs

Intraprocedural vs. Interprocedural Analysis

The MVP Solution

Procedure Flow Graphs

Procedure Flow Graphs I

Definition 17.2 (Procedure flow graphs; extends Def. 2.3 and 2.4)

The auxiliary functions **init**, **final**, and **flow** are extended as follows:

$$\text{init}(\text{proc } [P(\text{val } x, \text{res } y)]^{l_n} \text{ is } c \text{ [end]}^{l_x}) := l_n$$

$$\text{final}(\text{proc } [P(\text{val } x, \text{res } y)]^{l_n} \text{ is } c \text{ [end]}^{l_x}) := \{l_x\}$$

$$\text{flow}(\text{proc } [P(\text{val } x, \text{res } y)]^{l_n} \text{ is } c \text{ [end]}^{l_x}) := \{(l_n, \text{init}(c))\} \cup \text{flow}(c) \\ \cup \{(l, l_x) \mid l \in \text{final}(c)\}$$

$$\text{init}([\text{call } P(a, x)]_{l_r}^{l_c}) := l_c$$

$$\text{final}([\text{call } P(a, x)]_{l_r}^{l_c}) := \{l_r\}$$

$$\text{flow}([\text{call } P(a, x)]_{l_r}^{l_c}) := \{(l_c; l_n), (l_x; l_r)\}$$

Procedure Flow Graphs

Procedure Flow Graphs I

Definition 17.2 (Procedure flow graphs; extends Def. 2.3 and 2.4)

The auxiliary functions **init**, **final**, and **flow** are extended as follows:

$$\begin{aligned}\text{init}(\text{proc } [P(\text{val } x, \text{res } y)]^{l_n} \text{ is } c \text{ [end]}^{l_x}) &:= l_n \\ \text{final}(\text{proc } [P(\text{val } x, \text{res } y)]^{l_n} \text{ is } c \text{ [end]}^{l_x}) &:= \{l_x\} \\ \text{flow}(\text{proc } [P(\text{val } x, \text{res } y)]^{l_n} \text{ is } c \text{ [end]}^{l_x}) &:= \{(l_n, \text{init}(c))\} \cup \text{flow}(c) \\ &\quad \cup \{(l, l_x) \mid l \in \text{final}(c)\} \\ \text{init}([\text{call } P(a, x)]_{l_r}^{l_c}) &:= l_c \\ \text{final}([\text{call } P(a, x)]_{l_r}^{l_c}) &:= \{l_r\} \\ \text{flow}([\text{call } P(a, x)]_{l_r}^{l_c}) &:= \{(l_c; l_n), (l_x; l_r)\}\end{aligned}$$

Moreover the **interprocedural flow** of a program p c is defined by

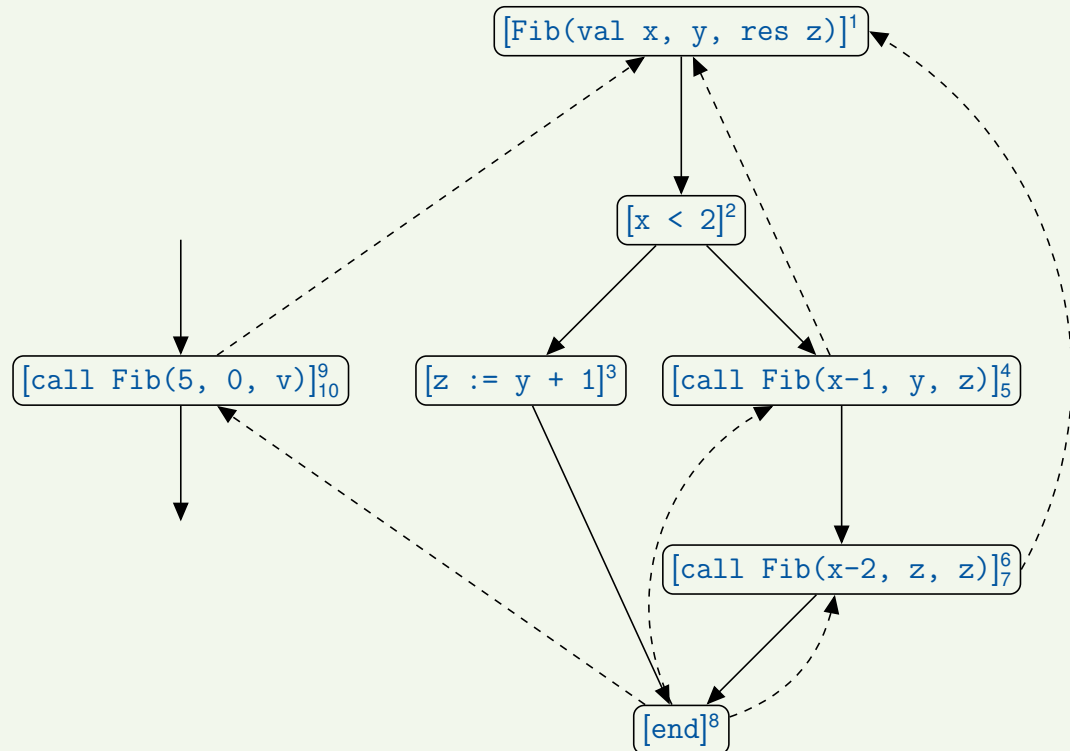
$$\begin{aligned}\text{iflow} &:= \{(l_c, l_n, l_x, l_r) \mid p \text{ contains } \text{proc } [P(\text{val } x, \text{res } y)]^{l_n} \text{ is } c \text{ [end]}^{l_x} \text{ and} \\ &\quad c \text{ contains } [\text{call } P(a, x)]_{l_r}^{l_c}\} \\ &\subseteq \text{Lab}^4\end{aligned}$$

Procedure Flow Graphs

Procedure Flow Graphs II

Example 17.3 (Fibonacci numbers)

```
proc [Fib(val x, y, res z)]1 is
  if [x < 2]2 then
    [z := y + 1]3
  else
    [call Fib(x-1, y, z)]45;
    [call Fib(x-2, z, z)]67
  end
[end]8;
[call Fib(5, 0, v)]910
```



Here iflow = $\{(9, 1, 8, 10), (4, 1, 8, 5), (6, 1, 8, 7)\}$ (dashed edges)

Procedure Flow Graphs

Procedure Flow Graphs and Dataflow Analysis

Interprocedural Dataflow Analysis

Data flow analysis uses static representation of programs to analyse summary information along paths

- by explicitly introducing **interprocedural paths** (MVP method) or
- by solving a **recursive equation system** (fixpoint method; later)

Procedure Flow Graphs

Procedure Flow Graphs and Dataflow Analysis

Interprocedural Dataflow Analysis

Data flow analysis uses static representation of programs to analyse summary information along paths

- by explicitly introducing **interprocedural paths** (MVP method) or
- by solving a **recursive equation system** (fixpoint method; later)

Important aspects

Soundness: All valid paths must be covered

- i.e, paths which represent legal control flow

Precision: Only valid paths should be covered

Efficiency: Only relevant valid paths should be covered

- i.e, paths which yield information that affects the summary information

Intraprocedural vs. Interprocedural Analysis

Outline of Lecture 17

Interprocedural Dataflow Analysis

Procedure Flow Graphs

Intraprocedural vs. Interprocedural Analysis

The MVP Solution

Intraprocedural vs. Interprocedural Analysis

Naive Formulation I

- **Attempt:** directly transfer techniques from intraprocedural analysis
⇒ treat $(l_c; l_n)$ like (l_c, l_n) and $(l_x; l_r)$ like (l_x, l_r)

Intraprocedural vs. Interprocedural Analysis

Naive Formulation I

- **Attempt:** directly transfer **techniques from intraprocedural analysis**
 \implies treat $(l_c; l_n)$ like (l_c, l_n) and $(l_x; l_r)$ like (l_x, l_r)
- Given: dataflow system $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$

Intraprocedural vs. Interprocedural Analysis

Naive Formulation I

- **Attempt:** directly transfer **techniques from intraprocedural analysis**
 \implies 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 $[call\ P(a, x)]_{l_r}^{l_c}$:
 transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \rightarrow D$ (definition later)
- For each procedure declaration $proc\ [P(val\ x, res\ y)]^n\ is\ c\ [end]^x$:
 transfer functions $\varphi_{l_n}, \varphi_{l_x} : D \rightarrow D$ (definition later)

Intraprocedural vs. Interprocedural Analysis

Naive Formulation I

- **Attempt:** directly transfer **techniques from intraprocedural analysis**
 \implies 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 $[call\ P(a, x)]_{l_r}^{l_c}$:
 transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \rightarrow D$ (definition later)
- For each procedure declaration $proc\ [P(val\ x, res\ y)]^{l_n}\ is\ c\ [end]^{l_x}$:
 transfer functions $\varphi_{l_n}, \varphi_{l_x} : D \rightarrow D$ (definition later)
- Induces **equation system**

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

Intraprocedural vs. Interprocedural Analysis

Naive Formulation I

- **Attempt:** directly transfer **techniques from intraprocedural analysis**
⇒ 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 $[call\ P(a, x)]_{l_r}^{l_c}$:
transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \rightarrow D$ (definition later)
- For each procedure declaration $proc\ [P(val\ x, res\ y)]^{l_n}\ is\ c\ [end]^{l_x}$:
transfer functions $\varphi_{l_n}, \varphi_{l_x} : D \rightarrow D$ (definition later)
- Induces **equation system**

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

- **Problem:** procedure calls $(l_c; l_n)$ and procedure returns $(l_x; l_r)$ treated like goto's
⇒ **nesting** of calls and returns ignored
⇒ **too many paths** considered
⇒ analysis information possibly **imprecise** (but still sound as all “valid” paths covered)

Intraprocedural vs. Interprocedural Analysis

Naive Formulation II

Example 17.4 (Fibonacci numbers)

```
proc [Fib(val x, y, res z)]1 is
  if [x < 2]2 then
    [z := y + 1]3
  else
    [call Fib(x-1, y, z)]45;
    [call Fib(x-2, z, z)]67
  end
[end]8;
[call Fib(5, 0, v)]910
```

Intraprocedural vs. Interprocedural Analysis

Naive Formulation II

Example 17.4 (Fibonacci numbers)

```
proc [Fib(val x, y, res z)]1 is
  if [x < 2]2 then
    [z := y + 1]3
  else
    [call Fib(x-1, y, z)]4;
    [call Fib(x-2, z, z)]6
  end
[end]8;
[call Fib(5, 0, v)]910
```

- Valid path: [9, 1, 2, 3, 8, 10]

Intraprocedural vs. Interprocedural Analysis

Naive Formulation II

Example 17.4 (Fibonacci numbers)

```
proc [Fib(val x, y, res z)]1 is
  if [x < 2]2 then
    [z := y + 1]3
  else
    [call Fib(x-1, y, z)]45;
    [call Fib(x-2, z, z)]67
  end
[end]8;
[call Fib(5, 0, v)]910
```

- Valid path: [9, 1, 2, 3, 8, 10]
- Invalid path: [9, 1, 2, 4, 1, 2, 3, 8, 10]

Intraprocedural vs. Interprocedural Analysis

Naive Formulation III

Example 17.5 (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;
  [y := y - 1]7
else
  [call P(2, y)]8;
  [y := y - 2]10
end;
[skip]11
```

Intraprocedural vs. Interprocedural Analysis

Naive Formulation III

Example 17.5 (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;
  [y := y - 1]7
else
  [call P(2, y)]8;
  [y := y - 2]10
end;
[skip]11
```

Two valid and two invalid paths:

- Valid: [4, 5, 1, 2, 3, 6, 7, 11]
⇒ $y = 0$ at label 11

Intraprocedural vs. Interprocedural Analysis

Naive Formulation III

Example 17.5 (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;
  [y := y - 1]7
else
  [call P(2, y)]8;
  [y := y - 2]10
end;
[skip]11
```

Two valid and two invalid paths:

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

Intraprocedural vs. Interprocedural Analysis

Naive Formulation III

Example 17.5 (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;
  [y := y - 1]7
else
  [call P(2, y)]8;
  [y := y - 2]10
end;
[skip]11
```

Two valid and two invalid paths:

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

Intraprocedural vs. Interprocedural Analysis

Naive Formulation III

Example 17.5 (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;
  [y := y - 1]7
else
  [call P(2, y)]8;
  [y := y - 2]10
end;
[skip]11
```

Two valid and two invalid paths:

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

Intraprocedural vs. Interprocedural Analysis

Naive Formulation III

Example 17.5 (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;
  [y := y - 1]7
else
  [call P(2, y)]8;
  [y := y - 2]10
end;
[skip]11
```

Two valid and two invalid paths:

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

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

The MVP Solution

Outline of Lecture 17

Interprocedural Dataflow Analysis

Procedure Flow Graphs

Intraprocedural vs. Interprocedural Analysis

The MVP Solution

The MVP Solution

Valid Paths I

- Consider only paths with **correct nesting** of procedure calls and returns
- Yields **MVP** solution (**Meet over all Valid Paths**)

Definition 17.6 (Valid path fragments)

Given a dataflow system $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$ and $l_1, l_2 \in Lab$, the set of **valid paths from l_1 to l_2** is generated by the nonterminal symbol $P[l_1, l_2]$ according to the following context-free grammar:

$$\begin{array}{ll} P[l_1, l_2] \rightarrow l_1 & \text{whenever } l_1 = l_2 \\ P[l_1, l_3] \rightarrow l_1, P[l_2, l_3] & \text{whenever } (l_1, l_2) \in F \\ P[l_c, l] \rightarrow l_c, P[l_n, l_x], P[l_r, l] & \text{whenever } (l_c, l_n, l_x, l_r) \in \text{iflow} \end{array}$$

The MVP Solution

Valid Paths II

Example 17.7 (Fibonacci numbers; cf. Example 17.4)

```
proc [Fib(val x, y, res z)]1 is
  if [x < 2]2 then
    [z := y + 1]3
  else
    [call Fib(x-1, y, z)]45;
    [call Fib(x-2, z, z)]67
  end
[end]8;
[call Fib(5, 0, v)]910
```

The MVP Solution

Valid Paths II

Example 17.7 (Fibonacci numbers; cf. Example 17.4)

```
proc [Fib(val x, y, res z)]1 is
  if [x < 2]2 then
    [z := y + 1]3
  else
    [call Fib(x-1, y, z)]45;
    [call Fib(x-2, z, z)]67
  end
[end]8;
[call Fib(5, 0, v)]910
```

Reminder:

$P[l_1, l_2] \rightarrow l_1$ for $l_1 = l_2$
 $P[l_1, l_3] \rightarrow l_1, P[l_2, l_3]$ for $(l_1, l_2) \in F$
 $P[l_c, l] \rightarrow l_c, P[l_n, l_x], P[l_r, l]$
for $(l_c, l_n, l_x, l_r) \in \text{iflow}$

The MVP Solution

Valid Paths II

Example 17.7 (Fibonacci numbers; cf. Example 17.4)

```
proc [Fib(val x, y, res z)]1 is
  if [x < 2]2 then
    [z := y + 1]3
  else
    [call Fib(x-1, y, z)]54;
    [call Fib(x-2, z, z)]76
  end
[end]8;
[call Fib(5, 0, v)]109
```

Reminder:

$P[l_1, l_2] \rightarrow l_1$ for $l_1 = l_2$
 $P[l_1, l_3] \rightarrow l_1, P[l_2, l_3]$ for $(l_1, l_2) \in F$
 $P[l_c, l] \rightarrow l_c, P[l_n, l_x], P[l_r, l]$
for $(l_c, l_n, l_x, l_r) \in \text{iflow}$

Valid paths from 9 to 10:

$P[9, 10] \rightarrow 9, P[1, 8], P[10, 10]$
 $P[1, 8] \rightarrow 1, P[2, 8]$
 $P[2, 8] \rightarrow 2, P[3, 8]$
 $P[2, 8] \rightarrow 2, P[4, 8]$
 $P[3, 8] \rightarrow 3, P[8, 8]$
 $P[4, 8] \rightarrow 4, P[1, 8], P[5, 8]$
 $P[5, 8] \rightarrow 5, P[6, 8]$
 $P[6, 8] \rightarrow 6, P[1, 8], P[7, 8]$
 $P[7, 8] \rightarrow 7, P[8, 8]$
 $P[8, 8] \rightarrow 8$
 $P[10, 10] \rightarrow 10$

The MVP Solution

Valid Paths II

Example 17.7 (Fibonacci numbers; cf. Example 17.4)

```
proc [Fib(val x, y, res z)]1 is
  if [x < 2]2 then
    [z := y + 1]3
  else
    [call Fib(x-1, y, z)]54;
    [call Fib(x-2, z, z)]76
  end
[end]8;
[call Fib(5, 0, v)]109
```

Reminder:

$P[l_1, l_2] \rightarrow l_1$ for $l_1 = l_2$
 $P[l_1, l_3] \rightarrow l_1, P[l_2, l_3]$ for $(l_1, l_2) \in F$
 $P[l_c, l] \rightarrow l_c, P[l_n, l_x], P[l_r, l]$
for $(l_c, l_n, l_x, l_r) \in \text{iflow}$

Valid paths from 9 to 10:

$P[9, 10] \rightarrow 9, P[1, 8], P[10, 10]$
 $P[1, 8] \rightarrow 1, P[2, 8]$
 $P[2, 8] \rightarrow 2, P[3, 8]$
 $P[2, 8] \rightarrow 2, P[4, 8]$
 $P[3, 8] \rightarrow 3, P[8, 8]$
 $P[4, 8] \rightarrow 4, P[1, 8], P[5, 8]$
 $P[5, 8] \rightarrow 5, P[6, 8]$
 $P[6, 8] \rightarrow 6, P[1, 8], P[7, 8]$
 $P[7, 8] \rightarrow 7, P[8, 8]$
 $P[8, 8] \rightarrow 8$
 $P[10, 10] \rightarrow 10$

Thus $[9, 1, 2, 3, 8, 10] \in L(P[9, 10])$,
 $[9, 1, 2, 4, 1, 2, 3, 8, 10] \notin L(P[9, 10])$

The MVP Solution

The MVP Solution I

Definition 17.8 (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

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

The MVP Solution

The MVP Solution I

Definition 17.8 (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

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

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

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

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

The MVP Solution

The MVP Solution II

Definition 17.9 (MVP solution)

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

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

where, for every $l \in Lab$,

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

The MVP Solution

The MVP Solution II

Definition 17.9 (MVP solution)

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

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

where, for every $l \in Lab$,

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

Corollary 17.10

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

The MVP Solution

The MVP Solution II

Definition 17.9 (MVP solution)

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

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

where, for every $l \in Lab$,

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

Corollary 17.10

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

Proof.

1. since $VPath(l) \subseteq Path(l)$ for every $l \in Lab$
2. as $\text{mvp}(S) = \text{mop}(S)$ in intraprocedural case and MOP solution undecidable (Thm. 6.6) \square