Static Program Analysis

Lecture 18: Interprocedural Dataflow Analysis I (MVP Solution)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://moves.rwth-aachen.de/teaching/ws-1415/spa/

Winter Semester 2014/15

Online Registration for Seminars and Practical Courses (Praktika) in Summer Term 2015

Who?

Students of:
- Master Courses
- Bachelor Informatik (~~Pro~~Seminar!)

Where?

www.graphics.rwth-aachen.de/apse

When?

14.01.2015 - 28.01.2015

Principles of Programming Languages

- Seminar in Theoretical CS in Summer Semester 2015
- Addresses several aspects of programming languages and systems
- Emphasis on how principles underpin practical applications
- 15 topics in 5 areas:
  1. Program Analysis (interprocedural analysis, abstraction refinement)
  2. Model Checking
  3. Probabilistic Systems
  4. Concurrent Systems
  5. Separation Logic
- http://moves.rwth-aachen.de/teaching/ss-15/popl/

Outline

Overview

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

Overview

- **So far:** only intraprocedural analyses (i.e., without user-defined functions or procedures or just within their bodies)
- **Now:** 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
- **Complications:**
  - correct matching between calls and returns
  - parameter passing (aliasing effects)

Overview

- **So far:** only intraprocedural analyses (i.e., without user-defined functions or procedures or just within their bodies)
- **Now:** interprocedural dataflow analysis
- **Complications:**
  - correct matching between calls and returns
  - parameter passing (aliasing effects)
- **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)

Extending the Syntax

Syntactic categories:

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

Extending the Syntax

Syntactic categories:

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

Context-free grammar:

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

Extending the Syntax

Syntactic categories:

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

Context-free grammar:

$$p ::= \texttt{proc}\ [P\texttt{(val}\ x\texttt{,res}\ y\texttt{)}]^{l_n}\ \texttt{is}\ c\ [\texttt{end}]^{l_x}\texttt{;}p \mid \varepsilon \in PDec$$

$$c ::= [\texttt{skip}]^l \mid [x\ \texttt{:=}\ a]^l \mid c_1\texttt{;}c_2 \mid \texttt{if}\ [b]^l\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \mid \texttt{while}\ [b]^l\ \texttt{do}\ c \mid [\texttt{call}\ P\texttt{(}a\texttt{,}x\texttt{)}]^{l_c}_{l_r} \in Cmd$$

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

An Example

Example 18.1 (Fibonacci numbers)

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

$$
\begin{array}{l}
\texttt{proc [Fib(val x, y, res z)]}^1\ \texttt{is} \\
\quad \texttt{if [x<2]}^2\ \texttt{then} \\
\qquad \texttt{[z := y+1]}^3 \\
\quad \texttt{else} \\
\qquad \texttt{[call Fib(x-1, y, z)]}^4_5\texttt{;} \\
\qquad \texttt{[call Fib(x-2, z, z)]}^6_7\texttt{;} \\
\texttt{[end]}^8\texttt{;} \\
\texttt{[call Fib(5, 0, v)]}^9_{10}
\end{array}
$$

Definition 18.2 (Procedure flow graphs; extends Def. 2.3 and 2.4)

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

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

$$
\begin{aligned}
\mathsf{init}([\texttt{call}\ P(a,x)]^{l_c}_{l_r}) &:= l_c \\
\mathsf{final}([\texttt{call}\ P(a,x)]^{l_c}_{l_r}) &:= \{l_r\} \\
\mathsf{flow}([\texttt{call}\ P(a,x)]^{l_c}_{l_r}) &:= \{(l_c; l_n), (l_x; l_r)\}
\end{aligned}
$$

if $\texttt{proc}\ [P(\texttt{val}\ x,\texttt{res}\ y)]^{l_n}\ \texttt{is}\ c\ [\mathsf{end}]^{l_x}$ is in p.

Procedure Flow Graphs I

Definition 18.2 (Procedure flow graphs; extends Def. 2.3 and 2.4)

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

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

$$\begin{aligned}
\mathsf{init}([\texttt{call}\ P(a,x)]^{l_c}_{l_r}) &:= l_c \\
\mathsf{final}([\texttt{call}\ P(a,x)]^{l_c}_{l_r}) &:= \{l_r\} \\
\mathsf{flow}([\texttt{call}\ P(a,x)]^{l_c}_{l_r}) &:= \{(l_c; l_n), (l_x; l_r)\}
\end{aligned}$$

if $\texttt{proc}\ [P(\texttt{val}\ x,\texttt{res}\ y)]^{l_n}\ \texttt{is}\ c\ [\mathsf{end}]^{l_x}$ is in p.

Moreover the interprocedural flow of a program $p\,c$ is defined by

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

Procedure Flow Graphs II

Example 18.3 (Fibonacci numbers)

Flow graph of

$$
\begin{array}{l}
\texttt{proc [Fib(val x, y, res z)]}^{1}\texttt{ is} \\
\quad \texttt{if [x<2]}^{2}\texttt{ then} \\
\qquad \texttt{[z := y+1]}^{3} \\
\quad \texttt{else} \\
\qquad \texttt{[call Fib(x-1, y, z)]}^{4}_{5}\texttt{;} \\
\qquad \texttt{[call Fib(x-2, z, z)]}^{6}_{7}\texttt{;} \\
\texttt{[end]}^{8}\texttt{;} \\
\texttt{[call Fib(5, 0, v)]}^{9}_{10}
\end{array}
$$

(on the board)

Example 18.3 (Fibonacci numbers)

Flow graph of

$$
\begin{array}{l}
\texttt{proc [Fib(val x, y, res z)]}^1\ \texttt{is} \\
\quad \texttt{if [x<2]}^2\ \texttt{then} \\
\qquad \texttt{[z := y+1]}^3 \\
\quad \texttt{else} \\
\qquad \texttt{[call Fib(x-1, y, z)]}^4_5\texttt{;} \\
\qquad \texttt{[call Fib(x-2, z, z)]}^6_7\texttt{;} \\
\texttt{[end]}^8\texttt{;} \\
\texttt{[call Fib(5, 0, v)]}^9_{10}
\end{array}
$$

(on the board)

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

Outline

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)

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 $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$

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

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

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

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 $[\texttt{call}\ P(a,x)]_{l_r}^{l_c}$:
  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 imprecise (but still correct)

Naive Formulation II

Example 18.4 (Fibonacci numbers)

$\texttt{proc}\ [\texttt{Fib(val x, y, res z)}]^1\ \texttt{is}$
$\quad\texttt{if}\ [\texttt{x<2}]^2\ \texttt{then}$
$\qquad[\texttt{z := y+1}]^3$
$\quad\texttt{else}$
$\qquad[\texttt{call Fib(x-1, y, z)}]^4_5\texttt{;}$
$\qquad[\texttt{call Fib(x-2, z, z)}]^6_7\texttt{;}$
$[\texttt{end}]^8\texttt{;}$
$[\texttt{call Fib(5, 0, v)}]^9_{10}$

Naive Formulation II

Example 18.4 (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_5;
    [call Fib(x-2, z, z)]^6_7;
[end]^8;
[call Fib(5, 0, v)]^9_10
```

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

Naive Formulation II

Example 18.4 (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_5;
    [call Fib(x-2, z, z)]^6_7;
[end]^8;
[call Fib(5, 0, v)]^9_10
```

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

Naive Formulation III

Example 18.5 (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_6;
  [y := y-1]^7
else
  [call P(2, y)]^8_9;
  [y := y-2]^10;
[skip]^11
```

Naive Formulation III

Example 18.5 (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)]_6^5;
  [y := y-1]^7
else
  [call P(2, y)]_9^8;
  [y := y-2]^10;
[skip]^11
```

Two "valid" and two "invalid" paths:

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

Naive Formulation III

Example 18.5 (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_6;
  [y := y-1]^7
else
  [call P(2, y)]^8_9;
  [y := y-2]^10;
[skip]^11
```

Two "valid" and two "invalid" paths:

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

Naive Formulation III

Example 18.5 (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_6;
  [y := y-1]^7
else
  [call P(2, y)]^8_9;
  [y := y-2]^10;
[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

Naive Formulation III

Example 18.5 (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_6;
  [y := y-1]^7
else
  [call P(2, y)]^8_9;
  [y := y-2]^10;
[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

Naive Formulation III

Example 18.5 (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)]_6^5;
  [y := y-1]^7
else
  [call P(2, y)]_9^8;
  [y := y-2]^10;
[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 $y = \top$

Outline

Valid Paths I

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

Definition 18.6 (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 productions:

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

Valid Paths II

Example 18.7 (Fibonacci numbers; cf. Example 18.4)

```
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)]^4_5;
    [call Fib(x-2, z, z)]^6_7;
[end]^8;
[call Fib(5, 0, v)]^9_10
```

Example 18.7 (Fibonacci numbers; cf. Example 18.4)

```
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)]^4_5;
    [call Fib(x-2, z, z)]^6_7;
[end]^8;
[call Fib(5, 0, v)]^9_10
```

Reminder:

$P[l_1, l_2] \to l_1$
for $l_1 = l_2$

$P[l_1, l_3] \to l_1, P[l_2, l_3]$
for $(l_1, l_2) \in F$

$P[l_c, l] \to l_c, P[l_n, l_x], P[l_r, l]$
for $(l_c, l_n, l_x, l_r) \in \text{iflow}$

Valid Paths II

Example 18.7 (Fibonacci numbers; cf. Example 18.4)

```
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)]^4_5;
    [call Fib(x-2, z, z)]^6_7;
[end]^8;
[call Fib(5, 0, v)]^9_10
```

Reminder:

$P[l_1, l_2] \to l_1$
for $l_1 = l_2$

$P[l_1, l_3] \to l_1, P[l_2, l_3]$
for $(l_1, l_2) \in F$

$P[l_c, l] \to l_c, P[l_n, l_x], P[l_r, l]$
for $(l_c, l_n, l_x, l_r) \in \text{iflow}$

Valid paths from 9 to 10:

$$
\begin{aligned}
P[9,10] &\to 9, P[1,8], P[10,10] \\
P[1,8] &\to 1, P[2,8] \\
P[2,8] &\to 2, P[3,8] \\
P[2,8] &\to 2, P[4,8] \\
P[3,8] &\to 3, P[8,8] \\
P[4,8] &\to 4, P[1,8], P[5,8] \\
P[5,8] &\to 5, P[6,8] \\
P[6,8] &\to 6, P[1,8], P[7,8] \\
P[7,8] &\to 7, P[8,8] \\
P[8,8] &\to 8 \\
P[10,10] &\to 10
\end{aligned}
$$

Valid Paths II

Example 18.7 (Fibonacci numbers; cf. Example 18.4)

```
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)]^4_5;
    [call Fib(x-2, z, z)]^6_7;
[end]^8;
[call Fib(5, 0, v)]^9_10
```

Reminder:

$$P[l_1, l_2] \to l_1 \quad \text{for } l_1 = l_2$$

$$P[l_1, l_3] \to l_1, P[l_2, l_3] \quad \text{for } (l_1, l_2) \in F$$

$$P[l_c, l] \to l_c, P[l_n, l_x], P[l_r, l] \quad \text{for } (l_c, l_n, l_x, l_r) \in \text{iflow}$$

Valid paths from 9 to 10:

$$\begin{aligned}
P[9,10] &\to 9, P[1,8], P[10,10] \\
P[1,8] &\to 1, P[2,8] \\
P[2,8] &\to 2, P[3,8] \\
P[2,8] &\to 2, P[4,8] \\
P[3,8] &\to 3, P[8,8] \\
P[4,8] &\to 4, P[1,8], P[5,8] \\
P[5,8] &\to 5, P[6,8] \\
P[6,8] &\to 6, P[1,8], P[7,8] \\
P[7,8] &\to 7, P[8,8] \\
P[8,8] &\to 8 \\
P[10,10] &\to 10
\end{aligned}$$

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 I

Definition 18.8 (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

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

Definition 18.8 (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

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

For a path $\pi = [l_1, \ldots, l_{k-1}] \in VPath(l)$, 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 18.9 (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 $l \in Lab$,

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

The MVP Solution II

Definition 18.9 (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 18.10

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

The MVP Solution II

Definition 18.9 (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 18.10

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. since $\mathrm{mvp}(S) = \mathrm{mop}(S)$ in intraprocedural case, and by undecidability of MOP solution (cf. Theorem 7.4) □

Outline

Making Context Explicit

- **Goal:** adapt fixpoint solution to avoid invalid paths

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)

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

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

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 18.11 (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 $(l_c, l_n, l_x, l_r) \in \mathit{iflow}$ (and $\varphi_l : D \to D$ otherwise).
The interprocedural extension of S is given by

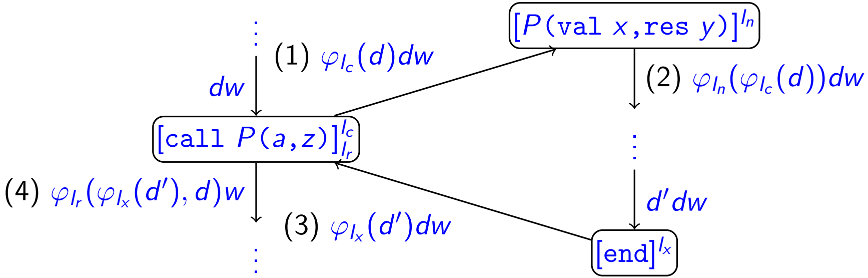
$$\hat{S} := (Lab, E, F, (\hat{D}, \hat{\sqsubseteq}), \hat{\iota}, \hat{\varphi})$$

where

- $\hat{D} := D^+$
- $d_1 \dots d_n \mathrel{\hat{\sqsubseteq}} d'_1 \dots d'_n$ 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 $l \in Lab \setminus \{l_c, l_r \mid (l_c, l_n, l_x, l_r) \in \mathit{iflow}\}$:
    $$\hat{\varphi}_l(dw) := \varphi_l(d)w$$
  - for each $(l_c, l_n, l_x, l_r) \in \mathit{iflow}$:
    $$\begin{aligned} \hat{\varphi}_{l_c}(dw) &:= \varphi_{l_c}(d)dw \\ \hat{\varphi}_{l_r}(d'dw) &:= \varphi_{l_r}(d', d)w \end{aligned}$$

Visualization of

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



The Interprocedural Extension III

Example 18.12 (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 $l \in Lab \setminus \{l_c, l_n, l_x, l_r \mid (l_c, l_n, l_x, l_r) \in \text{iflow}\}$,

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

The Interprocedural Extension III

Example 18.12 (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 $l \in Lab \setminus \{l_c, l_n, l_x, l_r \mid (l_c, l_n, l_x, l_r) \in \text{iflow}\}$,

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

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

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