Static Program Analysis

Lecture 19: Extensions I
(Interprocedural Dataflow Analysis – MVP Solution)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/spa11/

Summer Semester 2011

Outline

Overview

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

Overview

- **So far:** only intraprocedural analyses
  (i.e., without user-defined functions or procedures)
- **Now:** interprocedural dataflow analysis

Overview

- **So far:** only intraprocedural analyses
  (i.e., without user-defined functions or procedures)
- **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)
- **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	$PVar = \{\texttt{P}, \texttt{Q}, \ldots\}$	P
Procedure declarations	$PDec$	p
Commands (statements)	Cmd	c

Extending the Syntax

Syntactic categories:

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

Context-free grammar:

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

Extending the Syntax

Syntactic categories:

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

Context-free grammar:

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

- 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, second call-by-result

An Example

Example 19.1 (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)]^4_5;
    [call Fib(x-2, z, z)]^6_7;
[end]^8;
[call Fib(5, 0, v)]^9_10
```

Procedure Flow Graphs

Definition 19.2 (Procedure flow graphs)

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

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

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

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

Procedure Flow Graphs

Definition 19.2 (Procedure flow graphs)

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

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

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

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

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

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

Procedure Flow Graphs

Example 19.3 (Fibonacci numbers)

Flow graph of

```
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
```

(on the board)

Procedure Flow Graphs

Example 19.3 (Fibonacci numbers)

Flow graph of

```
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
```

(on the board)

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

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 $S = (L, E, F, (D, \sqsubseteq), \iota, \varphi)$
- For each procedure call $[\texttt{call}\ P\texttt{(}a\texttt{,}x\texttt{)}]^{l_c}_{l_r}$:
  transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \to D$ (definition later)
- For each procedure declaration $\texttt{proc}\ [P\texttt{(val}\ x\texttt{,res}\ y\texttt{)}]^{l_n}\ \texttt{is}\ c\ [\texttt{end}]^{l_x}$:
  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 $S = (L, E, F, (D, \sqsubseteq), \iota, \varphi)$
- For each procedure call $[\texttt{call}\ P\texttt{(}a\texttt{,}x\texttt{)}]^{l_c}_{l_r}$:
  transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \to D$ (definition later)
- For each procedure declaration $\texttt{proc}\ [P\texttt{(val}\ x\texttt{,res}\ y\texttt{)}]^{l_n}\ \texttt{is}\ c\ [\texttt{end}]^{l_x}$:
  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 $S = (L, E, F, (D, \sqsubseteq), \iota, \varphi)$
- For each procedure call $[\texttt{call } P\texttt{(}a\texttt{,}x\texttt{)}]^{l_c}_{l_r}$:
  transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \to D$ (definition later)
- For each procedure declaration $\texttt{proc } [P\texttt{(val } x\texttt{,res } y\texttt{)}]^{l_n} \texttt{ is } c\ [\texttt{end}]^{l_x}$:
  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
  $\Longrightarrow$ analysis information imprecise (but still correct)

Naive Formulation II

Example 19.4 (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
```

Naive Formulation II

Example 19.4 (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 19.4 (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 19.5 (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 19.5 (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

Naive Formulation III

Example 19.5 (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

Naive Formulation III

Example 19.5 (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 19.5 (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 19.5 (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

$\Longrightarrow$ actually $y = 0$ 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 19.6 (Valid paths I)

Given a dataflow system $S = (L, E, F, (D, \sqsubseteq), \iota, \varphi)$ and $l_1, l_2 \in L$, 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 productions:

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

Valid Paths II

Example 19.7 (Fibonacci numbers (cf. Example 19.4))

```
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
```

Valid Paths II

Example 19.7 (Fibonacci numbers (cf. Example 19.4))

```
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:

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

Valid Paths II

Example 19.7 (Fibonacci numbers (cf. Example 19.4))

```
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
```

Valid paths from 9 to 10:

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

Reminder:

$$
\begin{aligned}
P[l_1, l_2] &\to l_1 \quad \text{for } l_1 = l_2 \\
P[l_1, l_3] &\to l_1, P[l_2, l_3] \quad \text{for } (l_1, l_2) \in F \\
P[l_c, l] &\to l_c, P[l_n, l_x], P[l_r, l] \quad \text{for } (l_c, l_n, l_x, l_r) \in \text{iflow}
\end{aligned}
$$

Valid Paths II

Example 19.7 (Fibonacci numbers (cf. Example 19.4))

```
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:

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

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 19.8 (Valid paths II)

Let $S = (L, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system. For every $l \in L$, 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 $p = [l_1, \ldots, l_{k-1}] \in VPath(l)$, we define the transfer function $\varphi_p : D \to D$ by

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

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

The MVP Solution II

Definition 19.9 (MVP solution)

Let $S = (L, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system where $L = \{l_1, \ldots, l_n\}$. 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 $l \in L$,

$$\mathrm{mvp}(l) := \bigsqcup \{\varphi_p(\iota) \mid p \in \mathit{VPath}(l)\}.$$

The MVP Solution II

Definition 19.9 (MVP solution)

Let $S = (L, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system where $L = \{l_1, \ldots, l_n\}$. 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 $l \in L$,

$$\mathrm{mvp}(l) := \bigsqcup\{\varphi_p(\iota) \mid p \in \mathit{VPath}(l)\}.$$

Corollary 19.10

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

The MVP Solution II

Definition 19.9 (MVP solution)

Let $S = (L, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system where $L = \{l_1, \ldots, l_n\}$. 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 $l \in L$,

$$\mathrm{mvp}(l) := \bigsqcup\{\varphi_p(\iota) \mid p \in \mathit{VPath}(l)\}.$$

Corollary 19.10

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

Proof.

1. since $\mathit{VPath}(l) \subseteq \mathit{Path}(l)$ for every $l \in L$
2. since $\mathrm{mvp}(S) = \mathrm{mop}(S)$ in intraprocedural case, and by undecidability of MOP solution (cf. Theorem 6.2)

□