Static Program Analysis

Lecture 18: Interprocedural Dataflow Analysis I (MVP Solution)

Winter Semester 2016/17

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-1617/spa/
Online Registration for Seminars and Practical Courses (Praktika) in Summer Term 2017

Who?
Students of:  ▪ Master Courses
              ▪ Bachelor Informatik (ProSeminar!)

Where?
www.graphics.rwth-aachen.de/apse

When?
13.01.2017 – 29.01.2017
Seminar *Verification and Static Analysis of Software* (SS 2017)

### Topics

- Pointer and shape analysis
- Advanced model checking techniques
- Analysis of probabilistic programs
- ...

### More information

[https://moves.rwth-aachen.de/teaching/ss-17/vsas/](https://moves.rwth-aachen.de/teaching/ss-17/vsas/)

### Registration

between January 13 and 29 via

[https://www.graphics.rwth-aachen.de/apse/](https://www.graphics.rwth-aachen.de/apse/)
Interprocedural Dataflow Analysis

Outline of Lecture 18

Interprocedural Dataflow Analysis

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
- **Complications:**
  - correct matching between calls and returns
  - parameter passing (aliasing effects)

```
main(head, tail)
reverse(head, tail)
tmp := head
head := tail
tail := tmp
exit
```

```
reverse(cur, tail)
if (cur != tail)
tmp := cur.prev
cur.prev := cur.next
cur.next := tmp
reverse(cur.prev, tail)
if (cur = tail)
exit
```
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)
• **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)

```plaintext
main(head, tail)
reverse(head, tail)
tmp := head
head := tail
tail := tmp
exit
```

```plaintext
reverse(cur, tail)
if (cur != tail)
  tmp := cur.prev
  cur.prev := cur.next
  cur.next := tmp
reverse(cur.prev, tail)
```

```plaintext
if (cur = tail)
  exit
```
### Interprocedural Dataflow Analysis

#### Extending the Syntax

**Syntactic categories:**

<table>
<thead>
<tr>
<th>Category</th>
<th>Domain</th>
<th>Meta variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure identifiers</td>
<td>$\text{Pid} = {P, Q, \ldots}$</td>
<td>$P$</td>
</tr>
<tr>
<td>Procedure declarations</td>
<td>$P\text{Dec}$</td>
<td>$p$</td>
</tr>
<tr>
<td>Commands (statements)</td>
<td>$\text{Cmd}$</td>
<td>$c$</td>
</tr>
</tbody>
</table>

- All labels and procedure names in program $p$ are distinct.
- In $\text{proc} [P \text{val} x \text{res} y] \text{ln is } c \text{end } lx$, \text{ln/}lx refers to the entry/exit of $P$.
- In $\text{call} P(a, x)$, \text{lc/lr} refers to the call/return from $P$. First parameter call-by-value (input), second call-by-result (output).
Interprocedural Dataflow Analysis

Extending the Syntax

Syntactic categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Domain</th>
<th>Meta variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure identifiers</td>
<td>( Pid = {P, Q, \ldots} )</td>
<td>( P )</td>
</tr>
<tr>
<td>Procedure declarations</td>
<td>( PDec )</td>
<td>( p )</td>
</tr>
<tr>
<td>Commands (statements)</td>
<td>( Cmd )</td>
<td>( c )</td>
</tr>
</tbody>
</table>

Context-free grammar:

\[
p 
\colon= \text{proc } [P(\text{val } x, \text{res } y)]^n \text{ is } c \text{ [end]}^k; p \mid \varepsilon \in PDec
\]
\[
c 
\colon= [\text{skip}'] \mid [x := a'] \mid c_1; c_2 \mid \text{if } [b]' \text{ then } c_1 \text{ else } c_2 \text{ end } \mid \\
\text{while } [b]' \text{ do } c \text{ end } \mid [\text{call } P(a, x)]^l \in Cmd
\]
Interprocedural Dataflow Analysis

Extending the Syntax

Syntactic categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Domain</th>
<th>Meta variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure identifiers</td>
<td>( Pid = {P, Q, \ldots} )</td>
<td>( P )</td>
</tr>
<tr>
<td>Procedure declarations</td>
<td>( PDec )</td>
<td>( p )</td>
</tr>
<tr>
<td>Commands (statements)</td>
<td>( Cmd )</td>
<td>( c )</td>
</tr>
</tbody>
</table>

Context-free grammar:

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

- All labels and procedure names in program \( p \) distinct
- In \( \text{proc} \ [P(val \ x, res \ y)] \in p \), \( l_n / l_x \) refers to the entry / exit of \( P \)
- In \( \text{call} \ P(a, x) \in Cmd \), \( l_c / l_r \) 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 18.1 (Fibonacci numbers)
(with extension by multiple call-by-value parameters)

```plaintext
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)]5;
    end
[else]6
    [call Fib(5, 0, v)]7
```

```plaintext
1 of 18 Static Program Analysis
Winter Semester 2016/17
Lecture 18: Interprocedural Dataflow Analysis I (MVP Solution)
```
**Interprocedural Dataflow Analysis**

**Procedure Flow Graphs I**

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

The auxiliary functions \( \text{init}, \text{final}, \) and \( \text{flow} \) are extended as follows:

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

Moreover the interprocedural flow of a program \( p \) is defined by

\[
\text{iflow} := \{ (l_c, l_n, l_x, l_r) \mid \text{p contains proc } [P(\text{val } x, \text{res } y)]^{l_n} \text{ is } c [\text{end}]^{l_x} \text{ and } \text{c contains } \text{[call } P(a, x)]_{l_c}^{l_c} \} \subseteq \text{Lab} 4.
\]
Definition 18.2 (Procedure flow graphs; extends Def. 2.3 and 2.4)

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

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

\[
\begin{align*}
\text{init}([\text{call } P(a, x)]^c_{l_c}) & := l_c \\
\text{final}([\text{call } P(a, x)]^c_{l_r}) & := \{ l_r \} \\
\text{flow}([\text{call } P(a, x)]^c_{l_r}) & := \{ (l_c; l_n), (l_x; l_r) \}
\end{align*}
\]

Moreover the interprocedural flow of a program \( p c \) is defined by

\[
\text{iflow} := \{ (l_c, l_n, l_x, l_r) \mid p \text{ contains proc } [P(\text{val } x, \text{res } y)]^n \text{ is } c [\text{end}]^k \text{ and } c \text{ contains } [\text{call } P(a, x)]^c_{l_r} \}
\]

\( \subseteq \text{Lab}^4 \)
Example 18.3 (Fibonacci numbers)

Flow graph of

```plaintext
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)]⁹
```

(on the board)
Interprocedural Dataflow Analysis

Procedure Flow Graphs II

Example 18.3 (Fibonacci numbers)

Flow graph of

\[
\begin{align*}
\text{proc } \text{Fib}(\text{val } x, \text{ y, res } z) \text{ is} \\
\quad \text{if } [x < 2] \text{ then} \\
\quad \quad [z := y + 1] \\
\quad \text{else} \\
\quad \quad [\text{call Fib}(x-1, y, z)] \\
\quad \quad [\text{call Fib}(x-2, z, z)] \\
\quad \text{end} \\
\quad [\text{end}] \\
\quad [\text{call Fib}(5, 0, v)]
\end{align*}
\]

(on the board)

Here \( \text{iflow} = \{(9, 1, 8, 10), (4, 1, 8, 5), (6, 1, 8, 7)\} \)
Intraprocedural vs. Interprocedural Analysis

Outline of Lecture 18

Interprocedural Dataflow Analysis

Intraprocedural vs. Interprocedural Analysis

The MVP Solution
Intraprocedural vs. Interprocedural Analysis

Naive Formulation I

• **Attempt:** directly transfer techniques from intraprocedural analysis
  - Imply: treat \((l_c, l_n)\) like \((l_c, l_n)\) and \((l_x, l_r)\) like \((l_x, l_r)\)

  Given: dataflow system
  \(S = (Lab, E, F, (D, \sqsubseteq), \iota, \phi)\)

  For each procedure call \([call \ P(a, x)]\), transfer functions \(\phi_{lc}, \phi_{lr}\): \(D \rightarrow D\)

  For each procedure declaration \(proc \[P(val x, res y)]\), transfer functions \(\phi_{ln}, \phi_{lx}\): \(D \rightarrow D\)

Induces equation system

\[AI_{lc} = \begin{cases} \iota \text{ if } l \in E \sqsubseteq \{ \phi_{l'}(AI_{l'}) \mid (l', l) \in F \text{ or } (l'; l) \in F \} & \text{otherwise} \end{cases}\]

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

• **Attempt:** directly transfer techniques from intraprocedural analysis
  \[\implies \text{treat } (l_c; l_n) \text{ like } (l_c, l_n) \text{ and } (l_x; l_r) \text{ like } (l_x, l_r)\]

• Given: dataflow system \(S = (Lab, E, F, (D, ⊑), ι, ϕ)\)
Intraprocedural vs. Interprocedural Analysis

Naive Formulation I

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

- **Given:** dataflow system \( S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi) \)

- **For each procedure call** \([\text{call } P(a, x)]^l_c\):
  - transfer functions \( \varphi_{l_c}, \varphi_{l_r} : D \rightarrow D \) (definition later)

- **For each procedure declaration** \( \text{proc } [P(\text{val } x, \text{res } y)]^l_n \text{ is } c [\text{end}]^l_x\):
  - transfer functions \( \varphi_{l_n}, \varphi_{l_x} : D \rightarrow D \) (definition later)

- **Induces equation system**

  \[
  AI_I = \begin{cases} 
  l & \text{if } I \in E \\
  \bigsqcup \{ \varphi_{l'}(AI_{l'}) | (l', l) \in F \text{ or } (l'; I) \in F \} & \text{otherwise}
  \end{cases}
  \]
Intraprocedural vs. Interprocedural Analysis

Naive Formulation I

- **Attempt:** directly transfer techniques from intraprocedural analysis
  \[ \Rightarrow \text{treat } (l_c; l_n) \text{ like } (l_c, l_n) \text{ and } (l_x; l_r) \text{ like } (l_x, l_r) \]
- **Given:** dataflow system \( S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi) \)
- **For each procedure call** \([\text{call } P(a, x)]^k\):
  transfer functions \( \varphi_{l_c}, \varphi_{l_r} : D \rightarrow D \) (definition later)
- **For each procedure declaration** \( \text{proc } P(\text{val } x, \text{res } y)]^h \text{ is } c \text{ [end]}^l\):
  transfer functions \( \varphi_{l_n}, \varphi_{l_x} : D \rightarrow D \) (definition later)
- **Induces equation system**
  \[
  A_l = \begin{cases}
    \iota & \text{if } l \in E \\
    \sqcup \{ \varphi_{l'}(A_{l'}) | (l', l) \in F \text{ or } (l'; l) \in F \} & \text{otherwise}
  \end{cases}
  \]
- **Problem:** procedure calls \((l_c; l_n)\) and procedure returns \((l_x; l_r)\) treated like goto's
  \[ \Rightarrow \text{nesting of calls and returns ignored} \]
  \[ \Rightarrow \text{too many paths considered} \]
  \[ \Rightarrow \text{analysis information possibly imprecise (but still correct)} \]
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;
        [call Fib(x-2, z, z)]5;
    end
[end]8;
[call Fib(5, 0, v)]9
```
Intraprocedural vs. Interprocedural Analysis

Naive Formulation II

Example 18.4 (Fibonacci numbers)

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

• “Valid” path: [9, 1, 2, 3, 8, 10]
Intraprocedural vs. Interprocedural Analysis

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;
    [call Fib(x-2, z, z)]⁵
  end
[end]⁶;
[call Fib(5, 0, v)]⁷
```

- “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)

\[
\text{proc } [P(\text{val } x, \text{res } y)] \text{ is}
\begin{align*}
[y & := x] \\
[\text{end}] \\
\text{if } [y = 0] \text{ then}
\begin{align*}
[\text{call } P(1, y)] \\
[y & := y - 1]
\end{align*}
\text{else}
\begin{align*}
[\text{call } P(2, y)]
[y & := y - 2]
\end{align*}
\text{end;}
[\text{skip}]
\]
### Intraprocedural vs. Interprocedural Analysis

#### Naive Formulation III

**Example 18.5 (Impreciseness of constant propagation analysis)**

```plaintext
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]^6
else
  [call P(2, y)]^7;
  [y := y - 2]^8
end;
[skip]^9
```

Two “valid” and two “invalid” paths:

- **Valid:** \([4, 5, 1, 2, 3, 6, 7, 11]\)  
  \[\implies y = 0 \text{ at label } 11\]

- **Invalid:** \([4, 5, 1, 2, 3, 6, 7, 11]\)  
  \[\implies y = -1 \text{ at label } 11\]

- **Invalid:** \([4, 5, 1, 2, 3, 6, 7, 11]\)  
  \[\implies y = 1 \text{ at label } 11\]
### Naive Formulation III

**Example 18.5 (Impreciseness of constant propagation analysis)**

```plaintext
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]^6
else
    [call P(2, y)]^7
    [y := y - 2]^8
end;
[skip]^9
```

Two “valid” and two “invalid” paths:

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

但实际上总是y=0在11，但naive方法总是y=⊤。
Intraprocedural vs. Interprocedural Analysis

Naive Formulation III

Example 18.5 (Impreciseness of constant propagation analysis)

```plaintext
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]⁶
else
  [call P(2, y)]⁷;
  [y := y - 2]⁸
end;
[skip]¹¹
```

Two “valid” and two “invalid” paths:
- Valid: [4, 5, 1, 2, 3, 6, 7, 11]  
  \[\Rightarrow\]  y = 0 at label 11
- Valid: [4, 8, 1, 2, 3, 9, 10, 11]  
  \[\Rightarrow\]  y = 0 at label 11
- Invalid: [4, 5, 1, 2, 3, 9, 10, 11]  
  \[\Rightarrow\]  y = -1 at label 11
Intraprocedural vs. Interprocedural Analysis

Naive Formulation III

Example 18.5 (Impreciseness of constant propagation analysis)

```plaintext
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]⁶
else
  [call P(2, y)]⁷;
  [y := y - 2]⁸
end;
[skip]¹¹
```

Two “valid” and two “invalid” paths:

- **Valid:** [4, 5, 1, 2, 3, 6, 7, 11]  $\Rightarrow$  y = 0 at label 11
- **Valid:** [4, 8, 1, 2, 3, 9, 10, 11]  $\Rightarrow$  y = 0 at label 11
- **Invalid:** [4, 5, 1, 2, 3, 9, 10, 11]  $\Rightarrow$  y = -1 at label 11
- **Invalid:** [4, 8, 1, 2, 3, 6, 7, 11]  $\Rightarrow$  y = 1 at label 11
Intraprocedural vs. Interprocedural Analysis

Naive Formulation III

Example 18.5 (Impreciseness of constant propagation analysis)

```plaintext
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
⇒ actually always \( y = 0 \) at 11, but naive method yields \( y = \top \)
```

Two “valid” and two “invalid” paths:
- Valid: \([4, 5, 1, 2, 3, 6, 7, 11]\) \( \Rightarrow y = 0 \) at label 11
- Valid: \([4, 8, 1, 2, 3, 9, 10, 11]\) \( \Rightarrow y = 0 \) at label 11
- Invalid: \([4, 5, 1, 2, 3, 9, 10, 11]\) \( \Rightarrow y = -1 \) at label 11
- Invalid: \([4, 8, 1, 2, 3, 6, 7, 11]\) \( \Rightarrow y = 1 \) at label 11
Outline of Lecture 18

Interprocedural Dataflow Analysis

Intraprocedural vs. Interprocedural Analysis

The MVP Solution
The MVP Solution

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 \( S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi) \) and \( l_1, l_2 \in \text{Lab} \), 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:

\[
\begin{align*}
P[l_1, l_2] & \rightarrow l_1 \quad \text{whenever } l_1 = l_2 \\
P[l_1, l_3] & \rightarrow l_1, P[l_2, l_3] \quad \text{whenever } (l_1, l_2) \in F \\
P[l_c, l] & \rightarrow l_c, P[l_n, l_x], P[l_r, l] \quad \text{whenever } (l_c, l_n, l_x, l_r) \in \text{iflow}
\end{align*}
\]
Valid Paths II

Example 18.7 (Fibonacci numbers; cf. Example 18.4)

```
proc [Fib(val x, y, res z)] is
  if [x < 2] then
    [z := y + 1]
  else
    [call Fib(x-1, y, z)];
    [call Fib(x-2, z, z)]
  end
[end];
[call Fib(5, 0, v)]
```
Valid Paths II

Example 18.7 (Fibonacci numbers; cf. Example 18.4)

\[
\text{proc Fib(val } x, y, \text{ res } z) \text{ is}
\]
\[
\quad \text{if } x < 2 \text{ then }\]
\[
\quad \quad z := y + 1
\]
\[
\quad \text{else}\]
\[
\quad \quad \text{call Fib}(x-1, y, z)\]
\[
\quad \quad \text{call Fib}(x-2, z, z)
\]
\[
\quad \end
\]
\[
\quad \text{end}
\]
\[
\quad \text{call Fib}(5, 0, v)
\]

Reminder:

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

#### Example 18.7 (Fibonacci numbers; cf. Example 18.4)

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

#### Valid paths from 9 to 10:

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

---

**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 $(l_c, l_n, l_x, l_r) \in$ iflow
Example 18.7 (Fibonacci numbers; cf. Example 18.4)

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

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 \([9, 1, 2, 3, 8, 10] \in L(P[9, 10]), [9, 1, 2, 4, 1, 2, 3, 8, 10] \notin L(P[9, 10])\)
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 \( S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi) \) be a dataflow system. For every \( l \in \text{Lab} \), the set of valid paths up to \( l \) is given by

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

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

\[
\varphi_\pi := \varphi_{l_{k-1}} \circ \ldots \circ \varphi_{l_1} \circ \text{id}_D
\]

(so that \( \varphi[] = \text{id}_D \)).
The MVP Solution

The MVP Solution II

**Definition 18.9 (MVP solution)**

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

The **MVP solution** for \( S \) is determined by

\[
\text{mvp}(S) := (\text{mvp}(l_1), \ldots, \text{mvp}(l_n)) \in D^n
\]

where, for every \( l \in \text{Lab} \),

\[
\text{mvp}(l) := \bigsqcup_{\pi \in \text{VPath}(l)} \varphi_{\pi}(l).
\]
The MVP Solution

The MVP Solution II

Definition 18.9 (MVP solution)

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

\[
\text{mvp}(S) := (\text{mvp}(l_1), \ldots, \text{mvp}(l_n)) \in D^n
\]

where, for every \( l \in \text{Lab} \),

\[
\text{mvp}(l) := \bigcup \{ \varphi_\pi(l) \mid \pi \in \text{VPath}(l) \}.
\]

Corollary 18.10

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

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

where, for every $l \in Lab$,

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

Corollary 18.10

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

Proof.

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