

Program Analysis (470)

Inter-procedural Analysis

Herbert Wiklicky
herbert@doc.ic.ac.uk

Department of Computing
Imperial College London

Spring 2015

Extending WHILE

A program, P_* , in the extended WHILE language has the form

begin D_* S_* **end**

where D_* is a sequence of procedure declarations.

$D ::= \text{proc } p(\text{val } x, \text{res } y) \text{ is } S \text{ end} \mid D D$

$S ::= \dots \mid \text{call } p(a, z)$

or introducing labels:

$D ::= \text{proc } p(\text{val } x, \text{res } y) \text{ is}^{\ell_n} S \text{ end}^{\ell_x} \mid D D$

$S ::= \dots \mid [\text{call } p(a, z)]_{\ell_r}^{\ell_c}$

Extending WHILE

A program, P_* , in the extended WHILE language has the form

begin D_* S_* **end**

where D_* is a sequence of procedure declarations.

$D ::= \text{proc } p(\text{val } x, \text{res } y) \text{ is } S \text{ end} \mid D D$

$S ::= \dots \mid \text{call } p(a, z)$

or introducing labels:

$D ::= \text{proc } p(\text{val } x, \text{res } y) \text{ is}^{\ell_n} S \text{ end}^{\ell_x} \mid D D$

$S ::= \dots \mid [\text{call } p(a, z)]_{\ell_r}^{\ell_c}$

Extending WHILE

A program, P_* , in the extended WHILE language has the form

begin D_* S_* **end**

where D_* is a sequence of procedure declarations.

$D ::= \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is} \ S \ \mathbf{end} \mid D \ D$

$S ::= \dots \mid \mathbf{call} \ p(a, z)$

or introducing labels:

$D ::= \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \mid D \ D$

$S ::= \dots \mid [\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}$

Extending WHILE

A program, P_* , in the extended WHILE language has the form

begin D_* S_* **end**

where D_* is a sequence of procedure declarations.

$D ::= \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is} \ S \ \mathbf{end} \mid D \ D$

$S ::= \dots \mid \mathbf{call} \ p(a, z)$

or introducing labels:

$D ::= \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \mid D \ D$

$S ::= \dots \mid [\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}$

Fibonacci Example

begin

```
proc fib(val z, u, res v) is1  
  if [z<3]2 then [v := u+1]3  
    else ( [ call fib(z-1,u,v) ]4;  
           [ call fib(z-2,v,v) ]6 )  
  end8;  
  
  [ call fib(x,0,y) ]910
```

end

Extended Labelling: **call**

$$\begin{aligned} \mathit{init}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \ell_c \\ \mathit{final}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{\ell_r\} \\ \mathit{blocks}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{[\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}\} \\ \mathit{labels}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{\ell_c, \ell_r\} \\ \mathit{flow}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{(\ell_c; \ell_n), (\ell_x; \ell_r)\} \\ &\quad \text{if } \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \text{ is in } D_\star \end{aligned}$$

Extended Labelling: **call**

$$\begin{aligned} \mathit{init}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \ell_c \\ \mathit{final}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{\ell_r\} \\ \mathit{blocks}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{[\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}\} \\ \mathit{labels}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{\ell_c, \ell_r\} \\ \mathit{flow}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{(\ell_c; \ell_n), (\ell_x; \ell_r)\} \\ &\quad \text{if } \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \text{ is in } D_\star \end{aligned}$$

Extended Labelling: **call**

$$\begin{aligned} \mathit{init}([\mathbf{call} \ p(a, z)]_{l_r}^{l_c}) &= l_c \\ \mathit{final}([\mathbf{call} \ p(a, z)]_{l_r}^{l_c}) &= \{l_r\} \\ \mathit{blocks}([\mathbf{call} \ p(a, z)]_{l_r}^{l_c}) &= \{[\mathbf{call} \ p(a, z)]_{l_r}^{l_c}\} \\ \mathit{labels}([\mathbf{call} \ p(a, z)]_{l_r}^{l_c}) &= \{l_c, l_r\} \\ \mathit{flow}([\mathbf{call} \ p(a, z)]_{l_r}^{l_c}) &= \{(l_c; l_n), (l_x; l_r)\} \\ &\text{if } \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \mathbf{is}^{l_n} \ S \ \mathbf{end}^{l_x} \text{ is in } D_\star \end{aligned}$$

Extended Labelling: **call**

$$\begin{aligned} \mathit{init}([\mathbf{call} \ p(a, z)]_{l_r}^{l_c}) &= l_c \\ \mathit{final}([\mathbf{call} \ p(a, z)]_{l_r}^{l_c}) &= \{l_r\} \\ \mathit{blocks}([\mathbf{call} \ p(a, z)]_{l_r}^{l_c}) &= \{[\mathbf{call} \ p(a, z)]_{l_r}^{l_c}\} \\ \mathit{labels}([\mathbf{call} \ p(a, z)]_{l_r}^{l_c}) &= \{l_c, l_r\} \\ \mathit{flow}([\mathbf{call} \ p(a, z)]_{l_r}^{l_c}) &= \{(l_c; l_n), (l_x; l_r)\} \\ &\text{if } \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \mathbf{is}^{l_n} \ S \ \mathbf{end}^{l_x} \text{ is in } D_\star \end{aligned}$$

Extended Labelling: **call**

$$\begin{aligned} \mathit{init}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \ell_c \\ \mathit{final}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{\ell_r\} \\ \mathit{blocks}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{[\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}\} \\ \mathit{labels}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{\ell_c, \ell_r\} \\ \mathit{flow}([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) &= \{(\ell_c; \ell_n), (\ell_x; \ell_r)\} \\ &\text{if } \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \mathbf{is}^{\ell_n} \ \mathbf{S} \ \mathbf{end}^{\ell_x} \text{ is in } D_\star \end{aligned}$$

Inter-Procedural Flow

$(l_c; l_n)$ and $(l_x; l_r)$ are new kinds of flows:

- $(l_c; l_n)$ is the flow corresponding to **calling** a procedure at l_c and with l_n being the entry point for the procedure body, and
- $(l_x; l_r)$ is the flow corresponding to exiting a procedure body at l_x and **returning** to the call at l_r .

Inter-Procedural Flow

$(l_c; l_n)$ and $(l_x; l_r)$ are new kinds of flows:

- $(l_c; l_n)$ is the flow corresponding to **calling** a procedure at l_c and with l_n being the entry point for the procedure body, and
- $(l_x; l_r)$ is the flow corresponding to exiting a procedure body at l_x and **returning** to the call at l_r .

Inter-Procedural Flow

$(l_c; l_n)$ and $(l_x; l_r)$ are new kinds of flows:

- $(l_c; l_n)$ is the flow corresponding to **calling** a procedure at l_c and with l_n being the entry point for the procedure body, and
- $(l_x; l_r)$ is the flow corresponding to exiting a procedure body at l_x and **returning** to the call at l_r .

Extended Labelling: p

Next consider the program P_\star of the form

begin D_\star S_\star **end.**

For each procedure declaration

proc $p(\text{val } x, \text{res } y)$ **is** l_n S **end** l_x

we set:

$$\text{init}(p) = l_n$$

$$\text{final}(p) = \{l_x\}$$

$$\text{blocks}(p) = \{\mathbf{is}^{l_n}, \mathbf{end}^{l_x}\} \cup \text{blocks}(S)$$

$$\text{labels}(p) = \{l_n, l_x\} \cup \text{labels}(S)$$

$$\text{flow}(p) = \{(l_n, \text{init}(S))\} \cup \text{flow}(S) \cup \{(l, l_x) \mid l \in \text{final}(S)\}$$

Extended Labelling: p

Next consider the program P_* of the form

begin D_* S_* **end.**

For each procedure declaration

proc $p(\text{val } x, \text{res } y)$ **is** l_n S **end** l_x

we set:

$$\text{init}(p) = l_n$$

$$\text{final}(p) = \{l_x\}$$

$$\text{blocks}(p) = \{\text{is}^{l_n}, \text{end}^{l_x}\} \cup \text{blocks}(S)$$

$$\text{labels}(p) = \{l_n, l_x\} \cup \text{labels}(S)$$

$$\text{flow}(p) = \{(l_n, \text{init}(S))\} \cup \text{flow}(S) \cup \{(l, l_x) \mid l \in \text{final}(S)\}$$

Extended Labelling: p

Next consider the program P_* of the form

begin D_* S_* **end.**

For each procedure declaration

proc $p(\text{val } x, \text{res } y)$ **is** l_n S **end** l_x

we set:

$$\text{init}(p) = l_n$$

$$\text{final}(p) = \{l_x\}$$

$$\text{blocks}(p) = \{\text{is}^{l_n}, \text{end}^{l_x}\} \cup \text{blocks}(S)$$

$$\text{labels}(p) = \{l_n, l_x\} \cup \text{labels}(S)$$

$$\text{flow}(p) = \{(l_n, \text{init}(S))\} \cup \text{flow}(S) \cup \{(l, l_x) \mid l \in \text{final}(S)\}$$

Extended Labelling: p

Next consider the program P_\star of the form

begin D_\star S_\star **end.**

For each procedure declaration

proc $p(\text{val } x, \text{res } y)$ **is** l_n S **end** l_x

we set:

$$\text{init}(p) = l_n$$

$$\text{final}(p) = \{l_x\}$$

$$\text{blocks}(p) = \{\mathbf{is}^{l_n}, \mathbf{end}^{l_x}\} \cup \text{blocks}(S)$$

$$\text{labels}(p) = \{l_n, l_x\} \cup \text{labels}(S)$$

$$\text{flow}(p) = \{(l_n, \text{init}(S))\} \cup \text{flow}(S) \cup \{(l, l_x) \mid l \in \text{final}(S)\}$$

Extended Labelling: p

Next consider the program P_\star of the form

begin D_\star S_\star **end.**

For each procedure declaration

proc $p(\text{val } x, \text{res } y)$ **is** l_n S **end** l_x

we set:

$$\text{init}(p) = l_n$$

$$\text{final}(p) = \{l_x\}$$

$$\text{blocks}(p) = \{\mathbf{is}^{l_n}, \mathbf{end}^{l_x}\} \cup \text{blocks}(S)$$

$$\text{labels}(p) = \{l_n, l_x\} \cup \text{labels}(S)$$

$$\text{flow}(p) = \{(l_n, \text{init}(S))\} \cup \text{flow}(S) \cup \{(l, l_x) \mid l \in \text{final}(S)\}$$

Extended Labelling: p

Next consider the program P_* of the form

begin D_* S_* **end.**

For each procedure declaration

proc $p(\text{val } x, \text{res } y)$ **is** l_n S **end** l_x

we set:

$$\text{init}(p) = l_n$$

$$\text{final}(p) = \{l_x\}$$

$$\text{blocks}(p) = \{\mathbf{is}^{l_n}, \mathbf{end}^{l_x}\} \cup \text{blocks}(S)$$

$$\text{labels}(p) = \{l_n, l_x\} \cup \text{labels}(S)$$

$$\text{flow}(p) = \{(l_n, \text{init}(S))\} \cup \text{flow}(S) \cup \{(l, l_x) \mid l \in \text{final}(S)\}$$

Labelling: Program P_\star

For the entire program P_\star we set

$$init_\star = init(S_\star)$$

$$final_\star = final(S_\star)$$

$$blocks_\star = \bigcup \{ blocks(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup blocks(S_\star)$$

$$labels_\star = \bigcup \{ labels(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup labels(S_\star)$$

$$flow_\star = \bigcup \{ flow(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup flow(S_\star)$$

as well as $\mathbf{Lab}_\star = labels_\star$.

Labelling: Program P_\star

For the entire program P_\star we set

$$init_\star = init(S_\star)$$

$$final_\star = final(S_\star)$$

$$blocks_\star = \bigcup \{ blocks(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup blocks(S_\star)$$

$$labels_\star = \bigcup \{ labels(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup labels(S_\star)$$

$$flow_\star = \bigcup \{ flow(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup flow(S_\star)$$

as well as $\mathbf{Lab}_\star = labels_\star$.

Labelling: Program P_\star

For the entire program P_\star we set

$$init_\star = init(S_\star)$$

$$final_\star = final(S_\star)$$

$$blocks_\star = \bigcup \{ blocks(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup blocks(S_\star)$$

$$labels_\star = \bigcup \{ labels(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup labels(S_\star)$$

$$flow_\star = \bigcup \{ flow(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup flow(S_\star)$$

as well as $\mathbf{Lab}_\star = labels_\star$.

Labelling: Program P_\star

For the entire program P_\star we set

$$init_\star = init(S_\star)$$

$$final_\star = final(S_\star)$$

$$blocks_\star = \bigcup \{ blocks(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup blocks(S_\star)$$

$$labels_\star = \bigcup \{ labels(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup labels(S_\star)$$

$$flow_\star = \bigcup \{ flow(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup flow(S_\star)$$

as well as $\mathbf{Lab}_\star = labels_\star$.

Labelling: Program P_\star

For the entire program P_\star we set

$$init_\star = init(S_\star)$$

$$final_\star = final(S_\star)$$

$$blocks_\star = \bigcup \{ blocks(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup blocks(S_\star)$$

$$labels_\star = \bigcup \{ labels(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup labels(S_\star)$$

$$flow_\star = \bigcup \{ flow(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup flow(S_\star)$$

as well as $\mathbf{Lab}_\star = labels_\star$.

Labelling: Program P_\star

For the entire program P_\star we set

$$init_\star = init(S_\star)$$

$$final_\star = final(S_\star)$$

$$blocks_\star = \bigcup \{ blocks(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup blocks(S_\star)$$

$$labels_\star = \bigcup \{ labels(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup labels(S_\star)$$

$$flow_\star = \bigcup \{ flow(p) \mid \mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x} \\ \text{is in } D_\star \} \cup flow(S_\star)$$

as well as $\mathbf{Lab}_\star = labels_\star$.

Inter-procedural Flow

We shall also need to define a notion of **inter-procedural flow**

$$\begin{aligned} \textit{inter-flow}_\star &= \{(\ell_c, \ell_n, \ell_x, \ell_r) \mid P_\star \text{ contains} \\ &\quad [\mathbf{call } p(a, z)]_{\ell_r}^{\ell_c} \text{ as well as} \\ &\quad \mathbf{proc } p(\mathbf{val } x, \mathbf{res } y) \mathbf{is}^{\ell_n} S \mathbf{end}^{\ell_x}\} \end{aligned}$$

that clearly indicates the relationship between the labels of a procedure call and the corresponding procedure body.

Fibonacci: Labelling

For the Fibonacci program we have:

$$\begin{aligned} flow_{\star} = & \{(1, 2), (2, 3), (3, 8), \\ & (2, 4), (4; 1), (8; 5), (5, 6), (6; 1), (8; 7), (7, 8), \\ & (9; 1), (8; 10)\} \end{aligned}$$

$$inter-flow_{\star} = \{(9, 1, 8, 10), (4, 1, 8, 5), (6, 1, 8, 7)\}$$

$$init_{\star} = 9$$

$$final_{\star} = \{10\}$$

Fibonacci: Labelling

For the Fibonacci program we have:

$$\begin{aligned} \mathit{flow}_* &= \{(1, 2), (2, 3), (3, 8), \\ &\quad (2, 4), (4, 1), (8, 5), (5, 6), (6, 1), (8, 7), (7, 8), \\ &\quad (9, 1), (8, 10)\} \end{aligned}$$

$$\mathit{inter-flow}_* = \{(9, 1, 8, 10), (4, 1, 8, 5), (6, 1, 8, 7)\}$$

$$\mathit{init}_* = 9$$

$$\mathit{final}_* = \{10\}$$

Fibonacci: Labelling

For the Fibonacci program we have:

$$\begin{aligned} \mathit{flow}_* &= \{(1, 2), (2, 3), (3, 8), \\ &\quad (2, 4), (4, 1), (8; 5), (5, 6), (6; 1), (8; 7), (7, 8), \\ &\quad (9; 1), (8; 10)\} \end{aligned}$$

$$\mathit{inter-flow}_* = \{(9, 1, 8, 10), (4, 1, 8, 5), (6, 1, 8, 7)\}$$

$$\mathit{init}_* = 9$$

$$\mathit{final}_* = \{10\}$$

Fibonacci: Labelling

For the Fibonacci program we have:

$$\begin{aligned} \mathit{flow}_* &= \{(1, 2), (2, 3), (3, 8), \\ &\quad (2, 4), (4, 1), (8, 5), (5, 6), (6, 1), (8, 7), (7, 8), \\ &\quad (9, 1), (8, 10)\} \end{aligned}$$

$$\mathit{inter-flow}_* = \{(9, 1, 8, 10), (4, 1, 8, 5), (6, 1, 8, 7)\}$$

$$\mathit{init}_* = 9$$

$$\mathit{final}_* = \{10\}$$

Forward vs Backward Analysis

For a **forward analysis** we use:

- $F = flow_*$,
- $E = \{init_*\}$ and
- $IF = inter-flow_*$.

For a **backwards analysis** we use:

- $F = flow_*^R$,
- $E = final_*$ and
- $IF = inter-flow_*$.

Forward vs Backward Analysis

For a **forward analysis** we use:

- $F = flow_*$,
- $E = \{init_*\}$ and
- $IF = inter-flow_*$.

For a **backwards analysis** we use:

- $F = flow_*^R$,
- $E = final_*$ and
- $IF = inter-flow_*$.

Intra- vs Inter-Procedural Analysis

- for each procedure call

$$[\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}$$

we have two transfer functions f_{ℓ_c} and f_{ℓ_r} corresponding to calling the procedure and returning from the call, and

- for each procedure definition

$$\mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x}$$

we have two transfer functions f_{ℓ_n} and f_{ℓ_x} corresponding to entering and exiting the procedure body.

Intra- vs Inter-Procedural Analysis

- for each procedure call

$$[\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}$$

we have two transfer functions f_{ℓ_c} and f_{ℓ_r} corresponding to calling the procedure and returning from the call, and

- for each procedure definition

$$\mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \ \mathbf{is}^{\ell_n} \ S \ \mathbf{end}^{\ell_x}$$

we have two transfer functions f_{ℓ_n} and f_{ℓ_x} corresponding to entering and exiting the procedure body.

Inter-procedural Analysis

A simple (correct but quite imprecise) generalisation of the intraprocedural analysis treats interprocedural “jumps” $(\ell'; \ell)$ just like (ℓ', ℓ) .

$$A_{\bullet}(\ell) = f_{\ell}(A_{\circ}(\ell))$$

$$A_{\circ}(\ell) = \bigsqcup \{A_{\bullet}(\ell') \mid (\ell', \ell) \in F \text{ or } (\ell'; \ell) \in F\} \sqcup \iota_E^{\ell}$$

$$\iota_E^{\ell} = \begin{cases} \iota & \text{if } \ell \in E \\ \perp & \text{if } \ell \notin E \end{cases}$$

It is possible to exploit context information (e.g. using IF) in order to exclude “fictitious” executional paths.

Inter-procedural Analysis

A simple (correct but quite imprecise) generalisation of the intraprocedural analysis treats interprocedural “jumps” $(\ell'; \ell)$ just like (ℓ', ℓ) .

$$A_{\bullet}(\ell) = f_{\ell}(A_{\circ}(\ell))$$

$$A_{\circ}(\ell) = \bigsqcup \{A_{\bullet}(\ell') \mid (\ell', \ell) \in F \text{ or } (\ell'; \ell) \in F\} \sqcup \iota_E^{\ell}$$

$$\iota_E^{\ell} = \begin{cases} \iota & \text{if } \ell \in E \\ \perp & \text{if } \ell \notin E \end{cases}$$

It is possible to exploit context information (e.g. using $/F$) in order to exclude “fictitious” executional paths.

Inter-procedural Analysis

A simple (correct but quite imprecise) generalisation of the intraprocedural analysis treats interprocedural “jumps” $(\ell'; \ell)$ just like (ℓ', ℓ) .

$$A_{\bullet}(\ell) = f_{\ell}(A_{\circ}(\ell))$$

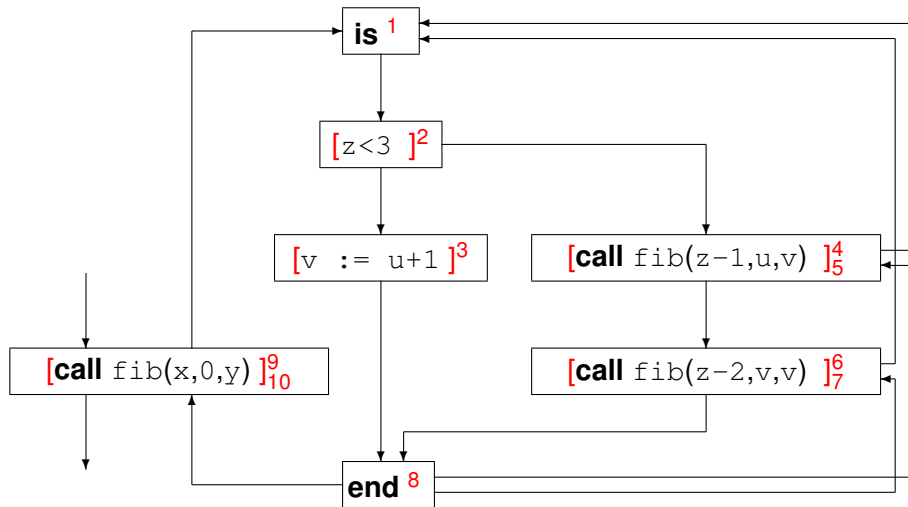
$$A_{\circ}(\ell) = \bigsqcup \{A_{\bullet}(\ell') \mid (\ell', \ell) \in F \text{ or } (\ell'; \ell) \in F\} \sqcup \iota_E^{\ell}$$

$$\iota_E^{\ell} = \begin{cases} \iota & \text{if } \ell \in E \\ \perp & \text{if } \ell \notin E \end{cases}$$

It is possible to exploit context information (e.g. using IF) in order to exclude “fictitious” executional paths.

Control-Flow for Fibonacci

```
proc fib(val z, u, res v)
```



Complete Paths

In order to construct a **Complete Path** CP_{l_1, l_2} between any two labels l_1 and l_2 in P_* in the same procedure we follow the following production rules:

$CP_{l_1, l_2} \longrightarrow l_1$ whenever $l_1 = l_2$

$CP_{l_1, l_3} \longrightarrow l_1, CP_{l_2, l_3}$ whenever $(l_1, l_2) \in F$
for a forward analysis this means that $(l_1, l_2) \in flow_*$

$CP_{l_c, l} \longrightarrow l_c, CP_{l_n, l_x}, CP_{l_r, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$
for a forward analysis this means that P_* contains $[call\ p(a, z)]_{l_r}^{l_c}$ and **proc** $p(val\ x, res\ y)$ **is** l_n **Send** l_x

Complete Paths

In order to construct a **Complete Path** CP_{l_1, l_2} between any two labels l_1 and l_2 in P_* in the same procedure we follow the following production rules:

$$CP_{l_1, l_2} \longrightarrow l_1$$

whenever $l_1 = l_2$

$$CP_{l_1, l_3} \longrightarrow l_1, CP_{l_2, l_3}$$

whenever $(l_1, l_2) \in F$
for a forward analysis this means that $(l_1, l_2) \in flow_*$

$$CP_{l_c, l} \longrightarrow l_c, CP_{l_n, l_x}, CP_{l_r, l}$$

whenever $(l_c, l_n, l_x, l_r) \in IF$
for a forward analysis this means that P_* contains $[\text{call } p(a, z)]_{l_r}^{l_c}$ and $\text{proc } p(\text{val } x, \text{res } y) \text{ is }^{l_n} \text{Send}^{l_x}$

Complete Paths

In order to construct a **Complete Path** CP_{l_1, l_2} between any two labels l_1 and l_2 in P_\star in the same procedure we follow the following production rules:

$CP_{l_1, l_2} \longrightarrow l_1$ whenever $l_1 = l_2$

$CP_{l_1, l_3} \longrightarrow l_1, CP_{l_2, l_3}$ whenever $(l_1, l_2) \in F$
for a forward analysis this means that $(l_1, l_2) \in flow_\star$

$CP_{l_c, l} \longrightarrow l_c, CP_{l_n, l_x}, CP_{l_r, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$
for a forward analysis this means that P_\star contains $[call\ p(a, z)]_{l_r}^{l_c}$ and $proc\ p(val\ x, res\ y)\ is^{l_n}\ Send^{l_x}$

Complete Paths

In order to construct a **Complete Path** CP_{l_1, l_2} between any two labels l_1 and l_2 in P_\star in the same procedure we follow the following production rules:

$CP_{l_1, l_2} \longrightarrow l_1$ whenever $l_1 = l_2$

$CP_{l_1, l_3} \longrightarrow l_1, CP_{l_2, l_3}$ whenever $(l_1, l_2) \in F$
for a forward analysis this means that $(l_1, l_2) \in flow_\star$

$CP_{l_c, l} \longrightarrow l_c, CP_{l_n, l_x}, CP_{l_r, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$
for a forward analysis this means that P_\star contains $[\mathbf{call} \ p(a, z)]_{l_r}^{l_c}$ and $\mathbf{proc} \ p(\mathbf{val} \ x, \mathbf{res} \ y) \mathbf{is}^{l_n} \ \mathbf{Send}^{l_x}$

Valid Paths

In order to construct a **Valid Path** in P_\star based on valid paths VP_{l_1, l_2} between any two labels l_1 and l_2 in P_\star we follow the following production rules:

$VP_\star \longrightarrow VP_{l_1, l_2}$ whenever $l_1 \in E \wedge l_2 \in \mathbf{Lab}_\star$

$VP_{l_1, l_2} \longrightarrow l_1$ whenever $l_1 = l_2$

$VP_{l_1, l_3} \longrightarrow l_1, VP_{l_2, l_3}$ whenever $(l_1, l_2) \in F$

$VP_{l_c, l} \longrightarrow l_c, CP_{l_n, l_x}, VP_{l_r, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$

$VP_{l_c, l} \longrightarrow l_c, VP_{l_n, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$

Valid Paths

In order to construct a **Valid Path** in P_\star based on valid paths VP_{l_1, l_2} between any two labels l_1 and l_2 in P_\star we follow the following production rules:

$VP_\star \longrightarrow VP_{l_1, l_2}$ whenever $l_1 \in E \wedge l_2 \in \mathbf{Lab}_\star$

$VP_{l_1, l_2} \longrightarrow l_1$ whenever $l_1 = l_2$

$VP_{l_1, l_3} \longrightarrow l_1, VP_{l_2, l_3}$ whenever $(l_1, l_2) \in F$

$VP_{l_c, l} \longrightarrow l_c, CP_{l_n, l_x}, VP_{l_r, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$

$VP_{l_c, l} \longrightarrow l_c, VP_{l_n, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$

Valid Paths

In order to construct a **Valid Path** in P_* based on valid paths VP_{l_1, l_2} between any two labels l_1 and l_2 in P_* we follow the following production rules:

$VP_* \longrightarrow VP_{l_1, l_2}$ whenever $l_1 \in E \wedge l_2 \in \mathbf{Lab}_*$

$VP_{l_1, l_2} \longrightarrow l_1$ whenever $l_1 = l_2$

$VP_{l_1, l_3} \longrightarrow l_1, VP_{l_2, l_3}$ whenever $(l_1, l_2) \in F$

$VP_{l_c, l} \longrightarrow l_c, CP_{l_n, l_x}, VP_{l_r, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$

$VP_{l_c, l} \longrightarrow l_c, VP_{l_n, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$

Valid Paths

In order to construct a **Valid Path** in P_* based on valid paths VP_{l_1, l_2} between any two labels l_1 and l_2 in P_* we follow the following production rules:

$VP_* \longrightarrow VP_{l_1, l_2}$ whenever $l_1 \in E \wedge l_2 \in \mathbf{Lab}_*$

$VP_{l_1, l_2} \longrightarrow l_1$ whenever $l_1 = l_2$

$VP_{l_1, l_3} \longrightarrow l_1, VP_{l_2, l_3}$ whenever $(l_1, l_2) \in F$

$VP_{l_c, l} \longrightarrow l_c, CP_{l_n, l_x}, VP_{l_r, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$

$VP_{l_c, l} \longrightarrow l_c, VP_{l_n, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$

Valid Paths

In order to construct a **Valid Path** in P_* based on valid paths VP_{l_1, l_2} between any two labels l_1 and l_2 in P_* we follow the following production rules:

$VP_* \longrightarrow VP_{l_1, l_2}$ whenever $l_1 \in E \wedge l_2 \in \mathbf{Lab}_*$

$VP_{l_1, l_2} \longrightarrow l_1$ whenever $l_1 = l_2$

$VP_{l_1, l_3} \longrightarrow l_1, VP_{l_2, l_3}$ whenever $(l_1, l_2) \in F$

$VP_{l_c, l} \longrightarrow l_c, CP_{l_n, l_x}, VP_{l_r, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$

$VP_{l_c, l} \longrightarrow l_c, VP_{l_n, l}$ whenever $(l_c, l_n, l_x, l_r) \in IF$

MVP Solution

$$\begin{aligned} \mathit{vpath}_o(\ell) = & \{ [l_1, \dots, l_{n-1}] \mid n \geq 1 \wedge l_n = \ell \\ & \wedge [l_1, \dots, l_n] \text{ is a valid path in } \{VP_\star\} \} \end{aligned}$$

$$\begin{aligned} \mathit{vpath}_\bullet(\ell) = & \{ [l_1, \dots, l_n] \mid n \geq 1 \wedge l_n = \ell \\ & \wedge [l_1, \dots, l_n] \text{ is a valid path in } \{VP_\star\} \} \end{aligned}$$

$$\mathit{MVP}_o(\ell) = \bigsqcup \{ f_{\vec{\ell}}(v) \mid \vec{\ell} \in \mathit{vpath}_o(\ell) \}$$

$$\mathit{MVP}_\bullet(\ell) = \bigsqcup \{ f_{\vec{\ell}}(v) \mid \vec{\ell} \in \mathit{vpath}_\bullet(\ell) \}$$

MVP Solution

$$\begin{aligned} \mathit{vpath}_o(\ell) = & \{ [l_1, \dots, l_{n-1}] \mid n \geq 1 \wedge l_n = \ell \\ & \wedge [l_1, \dots, l_n] \text{ is a valid path in } \{VP_\star\} \} \end{aligned}$$

$$\begin{aligned} \mathit{vpath}_\bullet(\ell) = & \{ [l_1, \dots, l_n] \mid n \geq 1 \wedge l_n = \ell \\ & \wedge [l_1, \dots, l_n] \text{ is a valid path in } \{VP_\star\} \} \end{aligned}$$

$$\mathit{MVP}_o(\ell) = \bigsqcup \{ f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in \mathit{vpath}_o(\ell) \}$$

$$\mathit{MVP}_\bullet(\ell) = \bigsqcup \{ f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in \mathit{vpath}_\bullet(\ell) \}$$

Flow-Sensitivity vs Flow-Insensitivity

All of the data flow analyses we have considered so far have been **flow-sensitive**: this just means that in general we would expect the analysis of a program $S_1; S_2$ to differ from the analysis of the program $S_2; S_1$ where the statements come in a different order.

Sometimes one considers **flow-insensitive** analyses where the order of statements is of no importance for the analysis being performed.

Clearly a flow-insensitive analysis may be much less precise than its flow-sensitive analogue but also it is likely to be much cheaper; since inter-procedural data flow analyses tend to be very costly, it is therefore useful to have a repertoire of techniques for reducing the cost.

Flow-Sensitivity vs Flow-Insensitivity

All of the data flow analyses we have considered so far have been **flow-sensitive**: this just means that in general we would expect the analysis of a program $S_1; S_2$ to differ from the analysis of the program $S_2; S_1$ where the statements come in a different order.

Sometimes one considers **flow-insensitive** analyses where the order of statements is of no importance for the analysis being performed.

Clearly a flow-insensitive analysis may be much less precise than its flow-sensitive analogue but also it is likely to be much cheaper; since inter-procedural data flow analyses tend to be very costly, it is therefore useful to have a repertoire of techniques for reducing the cost.

Flow-Sensitivity vs Flow-Insensitivity

All of the data flow analyses we have considered so far have been **flow-sensitive**: this just means that in general we would expect the analysis of a program $S_1; S_2$ to differ from the analysis of the program $S_2; S_1$ where the statements come in a different order.

Sometimes one considers **flow-insensitive** analyses where the order of statements is of no importance for the analysis being performed.

Clearly a flow-insensitive analysis may be much less precise than its flow-sensitive analogue but also it is likely to be much cheaper; since inter-procedural data flow analyses tend to be very costly, it is therefore useful to have a repertoire of techniques for reducing the cost.

Directly Assigned Variables

The set $IAV(S)$ of **directly assigned variables** gives for each statement S the set of variables that could be assigned in S – but ignoring the effect of procedure calls.

$$\begin{aligned} IAV([\text{skip}]^\ell) &= \emptyset \\ IAV([x := a]^\ell) &= \{x\} \\ IAV(S_1; S_2) &= IAV(S_1) \cup IAV(S_2) \\ IAV(\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2) &= IAV(S_1) \cup IAV(S_2) \\ IAV(\text{while } [b]^\ell \text{ do } S) &= IAV(S) \\ IAV([\text{call } p(a, z)]_{\ell_r}^{\ell_c}) &= \{z\} \end{aligned}$$

Directly Assigned Variables

The set $IAV(S)$ of **directly assigned variables** gives for each statement S the set of variables that could be assigned in S – but ignoring the effect of procedure calls.

$$\begin{aligned} IAV([\mathbf{skip}]^\ell) &= \emptyset \\ IAV([x := a]^\ell) &= \{x\} \\ IAV(S_1; S_2) &= IAV(S_1) \cup IAV(S_2) \\ IAV(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) &= IAV(S_1) \cup IAV(S_2) \\ IAV(\mathbf{while} [b]^\ell \mathbf{do} S) &= IAV(S) \\ IAV([\mathbf{call} p(a, z)]_{\ell_r}^{\ell_c}) &= \{z\} \end{aligned}$$

Directly Assigned Variables

The set $IAV(S)$ of **directly assigned variables** gives for each statement S the set of variables that could be assigned in S – but ignoring the effect of procedure calls.

$$IAV([\mathbf{skip}]^\ell) = \emptyset$$

$$IAV([x := a]^\ell) = \{x\}$$

$$IAV(S_1; S_2) = IAV(S_1) \cup IAV(S_2)$$

$$IAV(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) = IAV(S_1) \cup IAV(S_2)$$

$$IAV(\mathbf{while} [b]^\ell \mathbf{do} S) = IAV(S)$$

$$IAV([\mathbf{call} p(a, z)]_{\ell_r}^{\ell_c}) = \{z\}$$

Directly Assigned Variables

The set $IAV(S)$ of **directly assigned variables** gives for each statement S the set of variables that could be assigned in S – but ignoring the effect of procedure calls.

$$IAV([\mathbf{skip}]^\ell) = \emptyset$$

$$IAV([x := a]^\ell) = \{x\}$$

$$IAV(S_1; S_2) = IAV(S_1) \cup IAV(S_2)$$

$$IAV(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) = IAV(S_1) \cup IAV(S_2)$$

$$IAV(\mathbf{while} [b]^\ell \mathbf{do} S) = IAV(S)$$

$$IAV([\mathbf{call} p(a, z)]_{\ell_r}^{\ell_c}) = \{z\}$$

Directly Assigned Variables

The set $IAV(S)$ of **directly assigned variables** gives for each statement S the set of variables that could be assigned in S – but ignoring the effect of procedure calls.

$$\begin{aligned} IAV([\mathbf{skip}]^\ell) &= \emptyset \\ IAV([x := a]^\ell) &= \{x\} \\ IAV(S_1; S_2) &= IAV(S_1) \cup IAV(S_2) \\ IAV(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) &= IAV(S_1) \cup IAV(S_2) \\ IAV(\mathbf{while} [b]^\ell \mathbf{do} S) &= IAV(S) \\ IAV([\mathbf{call} p(a, z)]_{\ell_r}^{\ell_c}) &= \{z\} \end{aligned}$$

Directly Assigned Variables

The set $IAV(S)$ of **directly assigned variables** gives for each statement S the set of variables that could be assigned in S – but ignoring the effect of procedure calls.

$$\begin{aligned} IAV([\mathbf{skip}]^\ell) &= \emptyset \\ IAV([x := a]^\ell) &= \{x\} \\ IAV(S_1; S_2) &= IAV(S_1) \cup IAV(S_2) \\ IAV(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) &= IAV(S_1) \cup IAV(S_2) \\ IAV(\mathbf{while} [b]^\ell \mathbf{do} S) &= IAV(S) \\ IAV([\mathbf{call} \rho(a, z)]_{\ell_r}^{\ell_c}) &= \{z\} \end{aligned}$$

Directly Assigned Variables

The set $IAV(S)$ of **directly assigned variables** gives for each statement S the set of variables that could be assigned in S – but ignoring the effect of procedure calls.

$$\begin{aligned} IAV([\mathbf{skip}]^\ell) &= \emptyset \\ IAV([x := a]^\ell) &= \{x\} \\ IAV(S_1; S_2) &= IAV(S_1) \cup IAV(S_2) \\ IAV(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) &= IAV(S_1) \cup IAV(S_2) \\ IAV(\mathbf{while} [b]^\ell \mathbf{do} S) &= IAV(S) \\ IAV([\mathbf{call} p(a, z)]_{\ell_r}^{\ell_c}) &= \{z\} \end{aligned}$$

Immediately Called Procedures

The set $ICP(S)$ of **immediately called procedures** that gives for each statement S the set of procedure names that could be directly called in S – but ignoring the effect of procedure calls.

$$ICP([\mathbf{skip}]^\ell) = \emptyset$$

$$ICP([x := a]^\ell) = \emptyset$$

$$ICP(S_1; S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{while} [b]^\ell \mathbf{do} S) = ICP(S)$$

$$ICP([\mathbf{call} p(a, z)]_{\ell_r}^{\ell_c}) = \{p\}$$

Immediately Called Procedures

The set $ICP(S)$ of **immediately called procedures** that gives for each statement S the set of procedure names that could be directly called in S – but ignoring the effect of procedure calls.

$$ICP([\mathbf{skip}]^\ell) = \emptyset$$

$$ICP([x := a]^\ell) = \emptyset$$

$$ICP(S_1; S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{while} [b]^\ell \mathbf{do} S) = ICP(S)$$

$$ICP([\mathbf{call} p(a, z)]_{\ell_r}^{\ell_c}) = \{p\}$$

Immediately Called Procedures

The set $ICP(S)$ of **immediately called procedures** that gives for each statement S the set of procedure names that could be directly called in S – but ignoring the effect of procedure calls.

$$ICP([\mathbf{skip}]^\ell) = \emptyset$$

$$ICP([x := a]^\ell) = \emptyset$$

$$ICP(S_1; S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{while} [b]^\ell \mathbf{do} S) = ICP(S)$$

$$ICP([\mathbf{call} p(a, z)]_{\ell_r}^{\ell_c}) = \{p\}$$

Immediately Called Procedures

The set $ICP(S)$ of **immediately called procedures** that gives for each statement S the set of procedure names that could be directly called in S – but ignoring the effect of procedure calls.

$$ICP([\mathbf{skip}]^\ell) = \emptyset$$

$$ICP([x := a]^\ell) = \emptyset$$

$$ICP(S_1; S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{while} [b]^\ell \mathbf{do} S) = ICP(S)$$

$$ICP([\mathbf{call} p(a, z)]_{\ell_r}^{\ell_c}) = \{p\}$$

Immediately Called Procedures

The set $ICP(S)$ of **immediately called procedures** that gives for each statement S the set of procedure names that could be directly called in S – but ignoring the effect of procedure calls.

$$ICP([\mathbf{skip}]^\ell) = \emptyset$$

$$ICP([x := a]^\ell) = \emptyset$$

$$ICP(S_1; S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{while} [b]^\ell \mathbf{do} S) = ICP(S)$$

$$ICP([\mathbf{call} p(a, z)]_{\ell_r}^{\ell_c}) = \{p\}$$

Immediately Called Procedures

The set $ICP(S)$ of **immediately called procedures** that gives for each statement S the set of procedure names that could be directly called in S – but ignoring the effect of procedure calls.

$$ICP([\mathbf{skip}]^\ell) = \emptyset$$

$$ICP([x := a]^\ell) = \emptyset$$

$$ICP(S_1; S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{while} [b]^\ell \mathbf{do} S) = ICP(S)$$

$$ICP([\mathbf{call} \ p(a, z)]_{\ell_r}^{\ell_c}) = \{p\}$$

Immediately Called Procedures

The set $ICP(S)$ of **immediately called procedures** that gives for each statement S the set of procedure names that could be directly called in S – but ignoring the effect of procedure calls.

$$ICP([\mathbf{skip}]^\ell) = \emptyset$$

$$ICP([x := a]^\ell) = \emptyset$$

$$ICP(S_1; S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{if} [b]^\ell \mathbf{then} S_1 \mathbf{else} S_2) = ICP(S_1) \cup ICP(S_2)$$

$$ICP(\mathbf{while} [b]^\ell \mathbf{do} S) = ICP(S)$$

$$ICP([\mathbf{call} p(a, z)]_{\ell_r}^{\ell_c}) = \{p\}$$

Assigned Variables

The set $AV(p)$ of **assigned variables** in a procedure p is then given by the following (recursive) equation:

$$AV(p) = (IAV(S) \setminus \{x\}) \cup \bigcup \{AV(p') \mid p' \in ICP(S)\}$$

where the declaration

proc $p(\mathbf{val} \ x, \mathbf{res} \ y)$ **is** $^{\ell_n} S$ **end** $^{\ell_x}$

is in D_\star .

AV Example

```
begin
  proc fib(val z) is
    if z<3 then call add(1)
    else (call fib(z-1); call fib(z-2))
  end;
  proc add(val u) is
    (y:=y+u; u:=0)
  end;
  y:=0; call fib(x)
end
```

$$AV(\text{fib}) = (\emptyset \setminus \{z\}) \cup AV(\text{fib}) \cup AV(\text{add})$$

$$AV(\text{add}) = \{y, u\} \setminus \{u\}$$

AV Example

```
begin
  proc fib(val z) is
    if z<3 then call add(1)
    else (call fib(z-1); call fib(z-2))
  end;
  proc add(val u) is
    (y:=y+u; u:=0)
  end;
  y:=0; call fib(x)
end
```

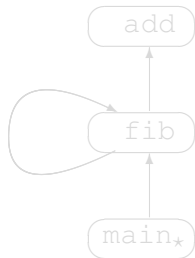
$$AV(\text{fib}) = (\emptyset \setminus \{z\}) \cup AV(\text{fib}) \cup AV(\text{add})$$

$$AV(\text{add}) = \{y, u\} \setminus \{u\}$$

AV Example: Call Relations

$$AV(\text{fib}) = (\emptyset \setminus \{z\}) \cup AV(\text{fib}) \cup AV(\text{add})$$

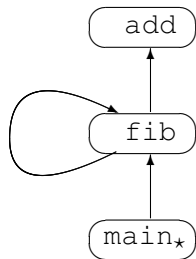
$$AV(\text{add}) = \{y, u\} \setminus \{u\}$$



AV Example: Call Relations

$$AV(\text{fib}) = (\emptyset \setminus \{z\}) \cup AV(\text{fib}) \cup AV(\text{add})$$

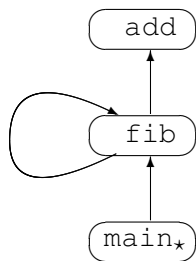
$$AV(\text{add}) = \{y, u\} \setminus \{u\}$$



AV Example: Call Relations

$$AV(\text{fib}) = (\emptyset \setminus \{z\}) \cup AV(\text{fib}) \cup AV(\text{add})$$

$$AV(\text{add}) = \{y, u\} \setminus \{u\}$$



The least solution to the equation system is

$$AV(\text{fib}) = AV(\text{add}) = \{y\}$$

i.e. only the variable y is assigned by the procedure calls.