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, $P_\star$, in the extended WHILE language has the form

$$\textbf{begin } D_\star \;\; S_\star \textbf{ end}$$

where $D_\star$ is a sequence of procedure declarations.

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

$$S ::= \cdots \mid \textbf{call } p(a, z)$$

or introducing labels:

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

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

Fibonacci Example

```
begin
        proc fib(val z, u, res v) is[1]
            if [z<3 ][2] then [v := u+1][3]
                        else ( [ call fib(z-1,u,v) ][4][5];
                               [ call fib(z-2,v,v) ][6][7] )
        end[8];

        [ call fib(x,0,y) ][9][10]
end
```


Extended Labelling: **call**

$$
\begin{aligned}
\mathit{init}([\textbf{call } p(a,z)]^{\ell_c}_{\ell_r}) &= \ell_c \\
\mathit{final}([\textbf{call } p(a,z)]^{\ell_c}_{\ell_r}) &= \{\ell_r\} \\
\mathit{blocks}([\textbf{call } p(a,z)]^{\ell_c}_{\ell_r}) &= \{[\textbf{call } p(a,z)]^{\ell_c}_{\ell_r}\} \\
\mathit{labels}([\textbf{call } p(a,z)]^{\ell_c}_{\ell_r}) &= \{\ell_c,\ell_r\} \\
\mathit{flow}([\textbf{call } p(a,z)]^{\ell_c}_{\ell_r}) &= \{(\ell_c;\ell_n),(\ell_x;\ell_r)\}
\end{aligned}
$$

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

Inter-Procedural Flow

$(\ell_c; \ell_n)$ and $(\ell_x; \ell_r)$ are new kinds of flows:

- $(\ell_c; \ell_n)$ is the flow corresponding to calling a procedure at ℓ_c and with ℓ_n being the entry point for the procedure body,and
- $(\ell_x; \ell_r)$ 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

$$\textbf{begin } D_\star\ S_\star \textbf{ end}.$$

For each procedure declaration

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

we set:

$$\begin{array}{rcl} \mathit{init}(p) & = & \ell_n \\ \mathit{final}(p) & = & \{\ell_x\} \\ \mathit{blocks}(p) & = & \{\textbf{is}^{\ell_n}, \textbf{end}^{\ell_x}\} \cup \mathit{blocks}(S) \\ \mathit{labels}(p) & = & \{\ell_n, \ell_x\} \cup \mathit{labels}(S) \\ \mathit{flow}(p) & = & \{(\ell_n, \mathit{init}(S))\} \cup \mathit{flow}(S) \cup \{(\ell, \ell_x) \mid \ell \in \mathit{final}(S)\} \end{array}$$

Labelling: Program $P_\star$

For the entire program $P_\star$ we set

$$\begin{array}{rcl}
init_\star & = & init(S_\star) \\
final_\star & = & final(S_\star) \\
blocks_\star & = & \bigcup\{blocks(p) \mid \textbf{proc}\ p(\textbf{val}\ x, \textbf{res}\ y)\ \textbf{is}^{\ell_n}\ S\ \textbf{end}^{\ell_x} \\
 & & \text{is in } D_\star\} \cup\ blocks(S_\star) \\
labels_\star & = & \bigcup\{labels(p) \mid \textbf{proc}\ p(\textbf{val}\ x, \textbf{res}\ y)\ \textbf{is}^{\ell_n}\ S\ \textbf{end}^{\ell_x} \\
 & & \text{is in } D_\star\} \cup\ labels(S_\star) \\
flow_\star & = & \bigcup\{flow(p) \mid \textbf{proc}\ p(\textbf{val}\ x, \textbf{res}\ y)\ \textbf{is}^{\ell_n}\ S\ \textbf{end}^{\ell_x} \\
 & & \text{is in } D_\star\} \cup\ flow(S_\star)
\end{array}$$

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

Inter-procedural Flow

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

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

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:

$$
\begin{aligned}
\mathit{flow}_\star &= \{(1,2),(2,3),(3,8),\\
&\quad (2,4),(4;1),(8;5),(5,6),(6;1),(8;7),(7,8),\\
&\quad (9;1),(8;10)\}\\
\mathit{inter\text{-}flow}_\star &= \{(9,1,8,10),(4,1,8,5),(6,1,8,7)\}\\
\mathit{init}_\star &= 9\\
\mathit{final}_\star &= \{10\}
\end{aligned}
$$

Forward vs Backward Analysis

For a forward analysis we use:

- $F = \mathit{flow}_\star$,
- $E = \{\mathit{init}_\star\}$ and
- $\mathit{IF} = \mathit{inter\text{-}flow}_\star$.

For a backwards analysis we use:

- $F = \mathit{flow}_\star^R$,
- $E = \mathit{final}_\star$ and
- $\mathit{IF} = \mathit{inter\text{-}flow}_\star$.

Intra- vs Inter-Procedural Analysis

- for each procedure call

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

  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

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

  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).

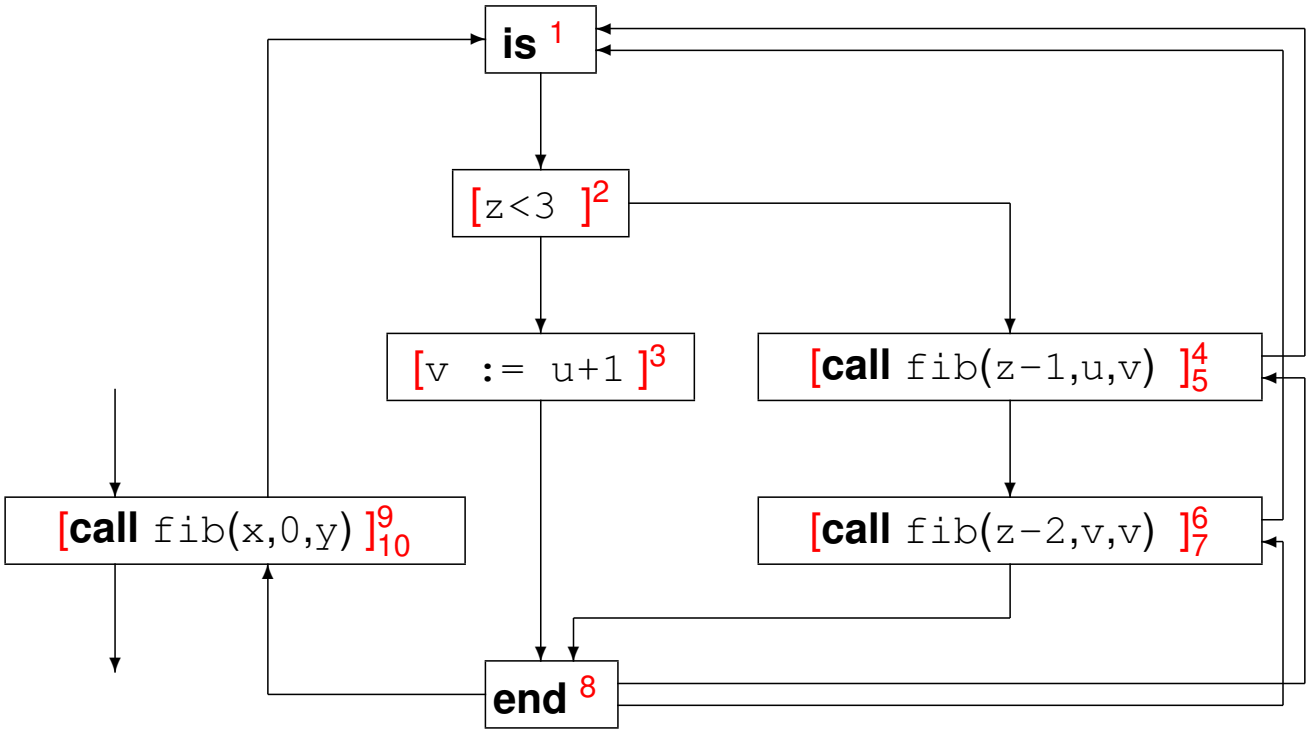
$$\begin{aligned} A_\bullet(\ell) &= f_\ell(A_\circ(\ell)) \\ A_\circ(\ell) &= \bigsqcup\{A_\bullet(\ell') \mid (\ell',\ell) \in F \text{ or } (\ell';\ell) \in F\} \sqcup \iota^\ell_E \end{aligned}$$

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

It is possible to exploit context information (e.g. using *IF*) in order to exclude "ficticious" 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 $P_\star$ in the same procedure we follow the following production rules:

$CP_{\ell_1,\ell_2} \longrightarrow \ell_1$	whenever $\ell_1 = \ell_2$
$CP_{\ell_1,\ell_3} \longrightarrow \ell_1, CP_{\ell_2,\ell_3}$	whenever $(\ell_1,\ell_2) \in F$ for a forward analysis this means that $(\ell_1,\ell_2) \in \mathit{flow}_\star$
$CP_{\ell_c,\ell} \longrightarrow \ell_c, CP_{\ell_n,\ell_x}, CP_{\ell_r,\ell}$	whenever $(\ell_c,\ell_n,\ell_x,\ell_r) \in \mathit{IF}$ for a forward analysis this means that $P_\star$ contains $[\textbf{call}\ p(a,z)]^{\ell_c}_{\ell_r}$ and $\textbf{proc}\ p(\textbf{val}\ x, \textbf{res}\ y)\ \textbf{is}^{\ell_n}\ S\ \textbf{end}^{\ell_x}$

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:

$VP_\star \longrightarrow VP_{\ell_1,\ell_2}$	whenever $\ell_1 \in E \ \wedge \ \ell_2 \in \mathbf{Lab}_\star$
$VP_{\ell_1,\ell_2} \longrightarrow \ell_1$	whenever $\ell_1 = \ell_2$
$VP_{\ell_1,\ell_3} \longrightarrow \ell_1, VP_{\ell_2,\ell_3}$	whenever $(\ell_1,\ell_2) \in F$
$VP_{\ell_c,\ell} \longrightarrow \ell_c, CP_{\ell_n,\ell_x}, VP_{\ell_r,\ell}$	whenever $(\ell_c,\ell_n,\ell_x,\ell_r) \in \mathit{IF}$
$VP_{\ell_c,\ell} \longrightarrow \ell_c, VP_{\ell_n,\ell}$	whenever $(\ell_c,\ell_n,\ell_x,\ell_r) \in \mathit{IF}$

MVP Solution

$$
\begin{aligned}
\mathit{vpath}_\circ(\ell) &= \{\, [\ell_1,\cdots,\ell_{n-1}] \mid n \geq 1 \ \wedge \ \ell_n = \ell \\
&\qquad \wedge\, [\ell_1,\cdots,\ell_n] \text{ is a valid path in } \{VP_\star\}\} \\
\mathit{vpath}_\bullet(\ell) &= \{\, [\ell_1,\cdots,\ell_n] \mid n \geq 1 \ \wedge \ \ell_n = \ell \\
&\qquad \wedge\, [\ell_1,\cdots,\ell_n] \text{ is a valid path in } \{VP_\star\}\}
\end{aligned}
$$

$$
\begin{aligned}
MVP_\circ(\ell) &= \bigsqcup \{ f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in \mathit{vpath}_\circ(\ell) \} \\
MVP_\bullet(\ell) &= \bigsqcup \{ f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in \mathit{vpath}_\bullet(\ell) \}
\end{aligned}
$$

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.

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

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.

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

Assigned Variables

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

$$
\begin{array}{rcl}
AV(p) & = & (IAV(S) \setminus \{x\}) \cup \\
 & & \bigcup \{AV(p') \mid p' \in ICP(S)\}
\end{array}
$$

where the declaration

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

is in $D_\star$.

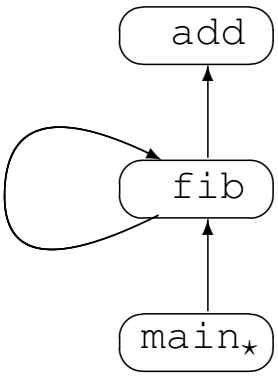
AV Example

```
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
```

$$\begin{aligned} AV(\texttt{fib}) &= (\emptyset \setminus \{\texttt{z}\}) \cup AV(\texttt{fib}) \cup AV(\texttt{add}) \\ AV(\texttt{add}) &= \{\texttt{y},\texttt{u}\} \setminus \{\texttt{u}\} \end{aligned}$$

AV Example: Call Relations

$$\begin{aligned} AV(\texttt{fib}) &= (\emptyset \setminus \{\texttt{z}\}) \cup AV(\texttt{fib}) \cup AV(\texttt{add}) \\ AV(\texttt{add}) &= \{\texttt{y},\texttt{u}\} \setminus \{\texttt{u}\} \end{aligned}$$



The least solution to the equation system is

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

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