Program Analysis
Lecture 13

Rayna Dimitrova

WS 2016/2017
Dates for the oral exam

If you have not, please fill the questionnaire for possible dates.
Possible student projects

Multiple possibilities for

▶ Bachelor thesis,
▶ Master thesis,
▶ internship projects

related to program analysis, at MPI-SWS.

If you are interested, contact

▶ Prof. Rupak Majumdar or
▶ Dr. Damien Zufferey
Possible student projects

Two interesting topics:

1. **Direct manipulation of procedurally generated objects.** We look at programs and programming languages that generate objects (procedural modeling) and try to add direct manipulation interface to these systems. One should be able not only to modify a program to make a new object, but also to modify the object and our system will figure out how to modify the program.

If you are interested in this topic, get in touch with Dr. Damien Zufferey (Room 315, Building G 26).
Two interesting topics:

2. **Programming and verification of fault-tolerant algorithms.** Fault-tolerant algorithms, like Paxos, are critical parts of many distributed systems and allows them to work in the presence of crashes and network failures. Our goal is to make it simpler to implement and verify fault-tolerant systems. In particular, we are looking a programming abstractions to decouple the logic of the algorithm from the fault-handling mechanism.

If you are interested in this topic, get in touch with Dr. Damien Zufferey (Room 315, Building G 26).
Recall: Intraprocedural dataflow analysis

A dataflow system is a tuple $S = (G, (D, \leq), i, \{f_b\}_{b \in B})$, where

- $G = (B, E, F)$ is a CFG
- $(D, \leq)$ is a complete lattice that satisfies (ACC)
- $i \in D$ is an initial value for extremal blocks
- $\{f_b\}_{b \in B}$ is a family of monotonic functions $f_b : D \to D$, one for each block in the CFG

The Join Over all Paths (JOP) solution of $S$ is

\[
JOP(S) := (X_{1}^{JOP}, \ldots, X_{|B|}^{JOP}),
\]

where for each block $b \in \{1, \ldots, |B|\},$

\[
X_{b}^{JOP} := \bigsqcup\{f_{\pi}(i) \mid \pi \in Paths(b)\}.
\]
Recall: Intraprocedural dataflow analysis

A **dataflow system** is a tuple \( S = (G, (D, \leq), i, \{ f_b \}_{b \in B}) \), where

- \( G = (B, E, F) \) is a CFG
- \( (D, \leq) \) is a complete lattice that satisfies (ACC)
- \( i \in D \) is an initial value for extremal blocks
- \( \{ f_b \}_{b \in B} \), is a family of monotonic functions \( f_b : D \to D \), one for each block in the CFG

The **least fixpoint (LFP)** solution approximates \( JOP \)

- associate with \( S \) a system of dataflow equations
- compute the least solution of the system of equations
Interprocedural analysis

Challenges

- (mutually) recursive procedures
- matching between calls and returns
- parameter passing (aliasing)
A simple programming languages with procedures

Simple setting:

- only top-level procedure declarations, no nested declarations
- one call-by-value and one call-by-result parameter
  (extension to multiple parameters is straightforward)

programs with procedures

\[
p ::= \text{proc} \left[ P(\text{val } x, \text{res } y) \right]^{l_n} \text{ is } c \left[ \text{end} \right]^{l_x} ; p \mid \varepsilon
\]

\[
c ::= [\text{skip}]^l \mid [x := a]^l \mid [\text{read}(x)]^l \mid [\text{print}(a)]^l \mid c_1 ; c_2
\]

\[
\quad \mid \text{if } [b]^l \text{ then } \{c_1\} \text{ else } \{c_2\}
\]

\[
\quad \mid \text{while } [b]^l \text{ do } \{c\}
\]

\[
\mid [\text{call } P(a, z)]^{l_c}_{l_r}
\]
A simple programming languages with procedures

\[
p ::= \text{proc } [P(\text{val } x, \text{res } y)]^{ln} \text{ is } c [\text{end}]^{lx}; p \mid \varepsilon
\]

\[
c ::= [\text{skip}]^l \mid [x := a]^l \mid [\text{read}(x)]^l \mid [\text{print}(a)]^l \mid c_1; c_2
\]

\[| \quad \text{if } [b]^l \text{ then } \{c_1\} \text{ else } \{c_2\}\]

\[| \quad \text{while } [b]^l \text{ do } \{c\}\]

\[| \quad [\text{call } P(a, z)]^{lc}_{lr}\]

- all labels and procedure names in program \( p; c \) are distinct
- \( ln \) is the entry of procedure \( P \), and \( lx \) is the exit of \( P \)
- \( lc \) refers to the call of \( P \), \( lr \) refers to the return from \( P \)
Example: a program with recursive procedures

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

\begin{itemize}
  \item \textbf{proc} Fib(val x, y, res z) 
    Fib: procedure name 
    x, y, z: formal parameters 
    local variables 
  \item recursive calls 
    \item \textbf{call} Fib(5, 0, v) 
      5, 0, v: actual parameters 
      v: global variable 
      result returned in v
\end{itemize}
Control-flow graph

For every procedure \texttt{proc} \[P(\text{val } x, \text{res } y)]^{l_n} \textbf{is } c' [\textbf{end}]^{l_x}\] we can construct a a control-flow graph \(G_P := (B_P, E_P, F_P)\), where \(E_P\) consist of the initial block with label \(l_n\) (in the case of forward analysis) or the final block \(l_x\) (in the case of backward analysis).

The interprocedural flow of a program \(p; c\) is defined by

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

Let \(G_c = (B_c, E_c, F_c)\) be the control-flow graph for the program \(c\).

We define the \textbf{control-flow graph} of \(p; c\) as the tuple

\[
G := (B_c \cup \bigcup_{P \text{ in } p} B_P, E_c \cup \bigcup_{P \text{ in } p} E_P, F_c \cup \bigcup_{P \text{ in } p} F_P, \text{iflow})
\]
Example: control-flow graph

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

iflow = {(9, 1, 8, 10), (4, 1, 8, 5), (6, 1, 8, 7)}
```

The red edges represent calls and returns, and are not part of $F$. 
Dataflow analysis for programs with procedures

We are given a dataflow system $S = (G, (D, \leq), i, \{f_l\}_{l \in \text{Labels}})$

- $G = (B, E, F, iflow)$ is the control flow graph
- $(D, \leq)$ is a complete lattice that satisfies (ACC)
- $i \in D$ is an initial value for extremal blocks in $E_c$
- $\{f_l\}_{l \in \text{Labels}}$, is a family of monotonic functions $f_l : D \to D$, one for each of the labels in the program $l \in \text{Labels}$
Dataflow analysis for programs with procedures

We are given a dataflow system $S = (G, (D, \leq), i, \{f_l\}_{l \in \text{Labels}})$

- for a block $[\ldots]^l$ in the intraprocedural fragment of the language, the transfer function $f_l$ is as defined as usual

- for a block $[P(\text{val } x, \text{res } y)]^{ln}$, the transfer function $f_{ln}$ is $id$

  $$f_{ln}(d) = d \text{ for all } d \in D$$

  The effect of procedure entry is handled at the procedure call.

- for a block $[\text{end}]^lx$, the transfer function $f_{lx}$ is $id$

  $$f_{lx}(d) = d \text{ for all } d \in D$$

  The effect of procedure exit is handled at the procedure return.
Dataflow analysis for programs with procedures

We are given a dataflow system \( S = (G, (D, \leq), i, \{f_l\}_{l \in \text{Labels}}) \)

- for a block \([\text{call } P(a, z)]^{lc}_{lr}\), we have two functions \(f_{lc}\) and \(f_{r}\).

We consider forward analysis and suppose that \(p; c\) contains \([\text{call } P(a, z)]^{lc}_{lr}\) and \(\text{proc } [P(\text{val } x, \text{res } y)]^{ln} \text{ is } c' \text{ [end]}^{lx}\).

The function

\[
f_{lc} : D \rightarrow D
\]

modifies the dataflow properties as required for passing to the procedure entry (set input parameter, reset output parameter).

The function

\[
f_{lr} : D \times D \rightarrow D
\]

modifies the dataflow properties as required for coming back from the procedure exit (reset parameters, set return value).
The transfer functions $f_{lr}$

$$f_{lr} : D \times D \to D$$

- first parameter: dataflow information at the call point, used to recover some of the information available before the call

- second parameter: dataflow information at the exit from the call

with proper context information, $f_{lc}$ and $f_{lr}$ can model the call stack

simple variation: completely ignore the information before the call
Dataflow analysis for programs with procedures

We are given a dataflow system \( S = (G, (D, \leq), i, \{f_l\}_{l \in \text{Labels}}) \).

**Question:** How to perform interprocedural dataflow analysis?

**Naive attempt:** directly use techniques from intraprocedural analysis

- for each \((l_c, l_n, l_x, l_r) \in \text{iflow}\), add edges \((l_c, l_n)\) and \((l_x, l_r)\)
Example: valid and invalid paths

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

valid path
[9, 1, 2, 3, 8, 10]

invalid path
[9, 1, 2, 4, 1, 2, 3, 8, 10]

Problem: nesting of calls and returns ignored
Example: invalid paths and analysis precision

```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;
}
[skip]¹¹
```

constant propagation analysis
valid path [4, 5, 1, 2, 3, 6, 7, 11]  
y = 0 at the entry of block 11
valid path [4, 8, 1, 2, 3, 9, 10, 11]  
y = 0 at the entry of block 11
invalid path [4, 5, 1, 2, 3, 9, 10, 11]  
y = -1 at the entry of block 11
invalid path [4, 8, 1, 2, 3, 6, 7, 11]  
y = 1 at the entry of block 11

at label 11 we have y = 0, but the naive analysis yields y = \top

Problem: considering too many paths results in imprecision
Context-free grammar defining valid paths

Consider only valid paths with correct nesting of calls and returns.

Given a control flow graph $G = (B, E, F, iflow)$, and labels $l_1$ and $l_2$, the set of valid paths from $l_1$ to $l_2$ is defined by the nonterminal symbol $P[l_1, l_2]$ according to the context-free grammar that has labels as terminals, nonterminals of the form $P[l, l']$ for each pair of labels $l$ and $l'$, and production rules

\[
\begin{align*}
P[l_1, l_2] & \rightarrow l_1 \quad \text{if } l_1 = l_2 \\
P[l_1, l_3] & \rightarrow l_1, P[l_2, l_3] \quad \text{if } (l_1, l_2) \in F \\
P[l_c, l] & \rightarrow l_c, P[l_n, l_x], P[l_r, l] \quad \text{if } (l_c, l_n, l_x, l_r) \in iflow
\end{align*}
\]
Example: valid paths

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

\[
P[l_1, l_2] \rightarrow l_1
\]
\[
\text{if } l_1 = l_2
\]
\[
P[l_1, l_3] \rightarrow l_1, P[l_2, l_3]
\]
\[
\text{if } (l_1, l_2) \in F
\]
\[
P[l_c, l] \rightarrow l_c, P[l_n, l_x], P[l_r, l]
\]
\[
\text{if } (l_c, l_n, l_x, l_r) \in iflow
\]

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

\[
[9, 1, 2, 3, 8, 10]
\]

generated by \( P[9, 10] \)

\[
[9, 1, 2, 4, 1, 2, 3, 8, 10]
\]

not generated by \( P[9, 10] \)
Consider a dataflow system $S = (G, (D, \leq), i, \{f_l\}_{l \in \text{Labels}})$ with control-flow graph $G = (B, E, F, \text{iflow})$.

The set of valid paths up to a label $l$ is

$$VPaths(l) = \{[l_1, \ldots, l_{k-1}] \mid k \geq 1, l_1 \text{ is initial}, l_k = l, [l_1, \ldots, l_k] \text{ is a valid path from } l_1 \text{ to } l_k\}$$

With a path $\pi = [l_1, \ldots, l_{k-1}]$ we associate the transfer function $f_\pi$ that takes into account procedure calls and returns.
Join Over all Valid Paths (JOVP) solution

Consider a dataflow system $S = (G, (D, \leq), i, \{f_l\}_{l \in \text{Labels}})$ with control-flow graph $G = (B, E, F, i\text{flow})$.

The JOVP solution for label $l$ (up to but not including $l$) is defined as

$$JOVP(l) := \bigsqcup \{f_\pi(i) \mid \pi \in \text{VPaths}(l)\}.$$ 

The JOVP solution of $S$ where $\{1, \ldots, n\}$ is the set of labels

$$JOVP(S) := (JOVP(1), \ldots, JOVP(n)).$$
Properties of the JOVP solution

For every dataflow system $S$ it holds that $JOVP(S) \leq JOP(S)$.

Proof: $VPaths(l) \subseteq Paths(l)$ for every label $l$.

The problem of computing the $JOVP$ solution is undecidable.

Proof: $JOVP(S) = JOP(S)$ in the intraprocedural case.
Methods for dataflow analysis

**Goal:** approximate the *JOVP* solution

**Approach:** compute least fixpoint solution where we avoid including too many invalid paths. We consider two methods:

1. **procedure summaries**
   - compute transfer functions for complete procedures
   - perform dataflow analysis using the procedure summaries

2. **including explicit call context**
   - encode the call history into the dataflow information
   - formulate dataflow equations corresponding to procedures
Analysis with procedure summaries

Consider $S = (G, (D, \leq), i, \{f_l\}_{l \in \text{Labels}})$ with $G = (B, E, F, \text{iflow})$.

Suppose that for each procedure $P$ in the program $p; c$ we have

$$f_P : D \rightarrow D$$

which summarises the effect of $\text{proc} \ [P(\text{val } x, \text{res } y)]^{l_n} \ \text{is } c' \ [\text{end}]^{l_x}$

$$f_P(d) = \bigsqcup \{f_{\pi}(d) \mid \pi \text{ is a valid path from } l_n \text{ to } l_x \} \text{ for all } d \in D$$

Then, we can define a system of dataflow equations using $f_P$.

**Problem:** the functions $f_P$ are usually not known.

**Approximation:** compute for each $P$ a function $f_P$ such that $f_P \geq f_\pi$ for all valid paths $\pi$ from $l_n$ to $l_x$. 
System of equations for procedure summaries

Consider $S = (G, (D, \leq), i, \{f_l\}_{l \in Labels})$ with $G = (B, E, F, iflow)$.

$(\hat{D}, \sqsubseteq)$ is the complete lattice of monotonic functions in $D \rightarrow D$ for every $f, g \in \hat{D}$, we have $f \sqsubseteq g$ iff $f(d) \leq g(d)$ for all $d \in D$.

We will compute $(f_P)_{P \in p}$ as the least solution in $\hat{D}^{\mid p\mid}$ of a system of equations ($\mid p\mid$ is the number of procedures in the program $p$; $c$).

**Note:** If the domain $D$ is finite, then each element of $\hat{D}$ can be easily represented effectively (for example, as a table). If $D$ is infinite, a symbolic representation of monotonic functions is necessary.
System of equations for procedure summaries

Consider $S = (G, (D, \leq), i, \{f_l\}_{l \in \text{Labels}})$ with $G = (B, E, F, \text{iflow})$.

The equations are over the following set of variables

- for each procedure $P$, a variable $Z_P$ denoting the summary of $P$
- for each block $b \in B$ that appears in the definition of procedure $\text{proc} \; [P(\text{val} \; x, \text{res} \; y)]^{l_n} \; \text{is} \; c' \; [\text{end}]^{l_x}$, a variable $Y_b$ denotes the effect of the procedure from $l_n$ up to the entry of block $b$

Thus, we define the following equations

- $Z_P = Y_{l_x}$, where $\text{proc} \; [P(\text{val} \; x, \text{res} \; y)]^{l_n} \; \text{is} \; c' \; [\text{end}]^{l_x}$ is in $p$
- for block $b$ in the definition of some procedure

$$Y_b = \begin{cases} id & \text{if } b = [P(\text{val} \; x, \text{res} \; y)]^{l_n}, \\ \bigsqcup \{g_{b'} \circ Y_{b'} \mid b' \in \text{pred}(b)\} & \text{otherwise} \end{cases}$$

**Question:** How do we define the functions $g_{b'} : D \to D$?
System of equations for procedure summaries

Consider \( S = (G, (D, \leq), i, \{f_l\}_{l \in \text{Labels}}) \) with \( G = (B, E, F, \text{iflow}) \).

Function \( g_{b'} \) for a block \( b' \).

- for blocks \([ \ldots ]^l \) in the intraprocedural fragment, and for blocks of the form \([P(\text{val } x, \text{res } y)]^l \) and \([\text{end}]^l \), \( g_b = f_l \) is given in \( S \)

- for blocks \( b' = [\text{call } P(a, z)]_{l_r}^{l_c} \)
  
  \[
  g_{b'} = \lambda d. \ f_{l_r}(d, Z_P \circ f_{l_c}(d))
  \]

  \( Z_P \circ f_{l_c} \) passes the dataflow information to the call of \( Z_P \)
  
  \( f_{l_r}(d, d') \) combines the result \( d' \) with the prior information \( d \)
Example: computing procedure summaries

\[
D = (\text{Vars} \to \{0, ?\}) \cup \{\bot\}
\]

\[
\top = (?, ?, ?), \ldots
\]

determine if variable is constant 0

procedure summary equations

\[
Y_1 = \text{id}
\]
\[
Y_2 = \text{id} \circ Y_1
\]
\[
Y_{3,4} = \text{id} \circ Y_2
\]
\[
Y_5 = \text{id} \circ Y_2
\]
\[
Y_6 = \text{assert}_{x,0} \circ Y_5
\]
\[
Y_7 = (\lambda d. \ f_4(d, \text{Zero} \circ f_3(d)) \circ Y_3
\]
\[
\quad \sqcup \text{set}_{y,x} \circ Y_6
\]
\[
\text{Z}_\text{Zero} = Y_7
\]
Example: computing procedure summaries

We represent function \( d = ( \text{Vars} \rightarrow \{0, ?\}) \) as \((d_x, d_y)\).

We define the call function \( f_3 \) such that \( f_3(d) = d[x \mapsto 0, y \mapsto ?] \).

We define the return function \( f_4 \) by \( f_4(d, d') = d[y \mapsto d'(y)] \).

Similarly

\[
\begin{align*}
f_8(d) &= d[x \mapsto 42, y \mapsto ?] \\
f_9(d, d') &= d[z \mapsto d'(y)]
\end{align*}
\]

We also use the following functions in \( D \rightarrow D \)

\[
\begin{align*}
\text{assert}_{x,0}((d_x, d_y, d_v)) &= (0, d_y, d_v) \\
\text{assert}_{y,0}((d_x, d_y, d_v)) &= (d_x, 0, d_v) \\
\text{set}_{y,x}((d_x, d_y, d_v)) &= (d_x, d_x, d_v) \\
\text{assert}_{x,y,0}((d_x, d_y, d_v)) &= (0, 0, d_v)
\end{align*}
\]

Note that \( \text{set}_{y,x} \circ \text{assert}_{x,0} = \text{assert}_{x,y,0} \).
Example: computing procedure summaries

The fixpoint computation of the procedure summaries is as follows

<table>
<thead>
<tr>
<th>$Y_1$</th>
<th>$Y_2$</th>
<th>$Y_{3,4}$</th>
<th>$Y_5$</th>
<th>$Y_6$</th>
<th>$Y_7$</th>
<th>$Z_{Zero}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>id</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>id</td>
<td>id</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>id</td>
<td>id</td>
<td>assert$_{x,0}$</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>id</td>
<td>id</td>
<td>assert$_{x,0}$</td>
<td>assert$_{x,y,0}$</td>
<td>⊥</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>id</td>
<td>id</td>
<td>assert$_{x,0}$</td>
<td>assert$_{x,y,0}$</td>
<td>assert$_{x,y,0}$</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>id</td>
<td>id</td>
<td>assert$_{x,0}$</td>
<td>assert$_{y,0}$</td>
<td>assert$_{x,y,0}$</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>id</td>
<td>id</td>
<td>assert$_{x,0}$</td>
<td>assert$_{y,0}$</td>
<td>assert$_{y,0}$</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>id</td>
<td>id</td>
<td>assert$_{x,0}$</td>
<td>assert$_{y,0}$</td>
<td>assert$_{y,0}$</td>
</tr>
</tbody>
</table>

Thus, the computed summary for procedure Zero is

$$f_{Zero}((d_x, d_y, d_v)) = (d_x, 0, d_v).$$
Least fixpoint solution of the procedure-summary equations

Consider $S = (G, (D, \leq), i, \{ f_l \}_{l \in \text{Labels}})$ with $G = (B, E, F, i\text{flow})$.

Consider the induced system of procedure-summary equations.

**Theorem**

If $(h_1, \ldots, h_{|B|}, f_1, \ldots, f_{|p|}) \in \widehat{D} |B| + |p|$ is the least solution (w.r.t. $\sqsubseteq |B| + |p|$) of the system of procedure-summary equations, then,

- if $b$ is a block in a procedure $P$, and $h_b$ is the function computed for $b$, then $h_b \sqsupseteq f_\pi$ for every valid path $\pi$ from the entry of the procedure $P$ to the entry point of block $b$.
- if $f_P$ is the function computed for a procedure $P$, then $f_P \sqsupseteq f_\pi$ for every valid path $\pi$ from the entry to the exit of $P$. 

25/25