

Static Program Analysis

Lecture 18: Interprocedural Dataflow Analysis I (MVP Solution)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)



noll@cs.rwth-aachen.de

<http://moves.rwth-aachen.de/teaching/ws-1415/spa/>

Winter Semester 2014/15

Online Registration for Seminars and Practical Courses (Praktika) in Summer Term 2015

Who?

- Students of:
- Master Courses
 - Bachelor Informatik (~~Pro~~Seminar!)

Where?

www.graphics.rwth-aachen.de/apse

When?

14.01.2015 - 28.01.2015

- Seminar in **Theoretical CS** in Summer Semester 2015
- Addresses several aspects of **programming languages and systems**
- Emphasis on how **principles** underpin practical applications
- 15 topics in 5 areas:
 - ① **Program Analysis** (interprocedural analysis, abstraction refinement)
 - ② Model Checking
 - ③ Probabilistic Systems
 - ④ Concurrent Systems
 - ⑤ Separation Logic
- <http://moves.rwth-aachen.de/teaching/ss-15/pop1/>

- 1 Interprocedural Dataflow Analysis
- 2 Intraprocedural vs. Interprocedural Analysis
- 3 The MVP Solution
- 4 The Interprocedural Fixpoint Solution

- **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)

Extending the Syntax

Syntactic categories:

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

Context-free grammar:

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

- All labels and procedure names in **program** $p c$ distinct
- In $\text{proc } [P(\text{val } x, \text{res } y)]^{l_n} \text{ is } c \text{ [end]}^{l_x}$, l_n/l_x refers to the **entry/exit** of P
- In $[\text{call } P(a, x)]_{l_r}^{l_c}$, l_c/l_r refers to the **call** of/**return** from P
- First parameter **call-by-value**, second **call-by-result**

Example 18.1 (Fibonacci numbers)

(with extension by multiple call-by-value parameters)

```
proc [Fib(val x, y, res z)]1 is
  if [x<2]2 then
    [z := y+1]3
  else
    [call Fib(x-1, y, z)]45;
    [call Fib(x-2, z, z)]67;
  [end]8;
[call Fib(5, 0, v)]910
```

Procedure Flow Graphs I

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

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

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

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

Moreover the **interprocedural flow** of a program pc is defined by

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

Example 18.3 (Fibonacci numbers)

Flow graph of

```
proc [Fib(val x, y, res z)]1 is
  if [x<2]2 then
    [z := y+1]3
  else
    [call Fib(x-1, y, z)]4;
    [call Fib(x-2, z, z)]6;
  [end]8;
[call Fib(5, 0, v)]910
```

(on the board)

Here $\text{iflow} = \{(9, 1, 8, 10), (4, 1, 8, 5), (6, 1, 8, 7)\}$

- 1 Interprocedural Dataflow Analysis
- 2 Intraprocedural vs. Interprocedural Analysis**
- 3 The MVP Solution
- 4 The Interprocedural Fixpoint Solution

Naive Formulation I

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

$$AI_l = \begin{cases} \iota & \text{if } l \in E \\ \bigsqcup \{ \varphi_{l'}(AI_{l'}) \mid (l', l) \in F \text{ or } (l; l') \in F \} & \text{otherwise} \end{cases}$$

- **Problem:** procedure calls $(l_c; l_n)$ and procedure returns $(l_x; l_r)$ treated like goto's
 - ⇒ **nesting** of calls and returns ignored
 - ⇒ too many **paths** considered
 - ⇒ analysis information **imprecise** (but still correct)

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)]6;
  [end]8;
[call Fib(5, 0, v)]910
```

- “Valid” path:
[9, 1, 2, 3, 8, 10]
- “Invalid” path:
[9, 1, 2, 4, 1, 2, 3, 8, 10]

Example 18.5 (Impreciseness of constant propagation analysis)

```
proc [P(val x, res y)]1 is
  [y := x]2
[end]3;
if [y=0]4 then
  [call P(1, y)]5;
  [y := y-1]7
else
  [call P(2, y)]8;
  [y := y-2]10;
[skip]11
```

Two “valid” and two “invalid” paths:

- Valid: [4, 5, 1, 2, 3, 6, 7, 11]
⇒ $y = 0$ at label 11
- Valid: [4, 8, 1, 2, 3, 9, 10, 11]
⇒ $y = 0$ at label 11
- Invalid: [4, 5, 1, 2, 3, 9, 10, 11]
⇒ $y = -1$ at label 11
- Invalid: [4, 8, 1, 2, 3, 6, 7, 11]
⇒ $y = 1$ at label 11

⇒ actually always $y = 0$ at 11, but naive method yields $y = \top$

- 1 Interprocedural Dataflow Analysis
- 2 Intraprocedural vs. Interprocedural Analysis
- 3 The MVP Solution
- 4 The Interprocedural Fixpoint Solution

- 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 $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$ and $l_1, l_2 \in Lab$, 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 productions:

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

Example 18.7 (Fibonacci numbers; cf. Example 18.4)

```
proc [Fib(val x, y, res z)]1 is
  if [x<2]2 then
    [z := y+1]3
  else
    [call Fib(x-1, y, z)]4;
    [call Fib(x-2, z, z)]6;
  [end]8;
[call Fib(5, 0, v)]910
```

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 $(l_c, l_n, l_x, l_r) \in \text{iflow}$

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

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

For a path $\pi = [l_1, \dots, l_{k-1}] \in VPath(l)$, we define the **transfer function** $\varphi_\pi : D \rightarrow D$ by

$$\varphi_\pi := \varphi_{l_{k-1}} \circ \dots \circ \varphi_{l_1} \circ \text{id}_D$$

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

Definition 18.9 (MVP solution)

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

$$\text{mvp}(S) := (\text{mvp}(l_1), \dots, \text{mvp}(l_n)) \in D^n$$

where, for every $l \in Lab$,

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

Corollary 18.10

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

Proof.

- 1 since $VPath(l) \subseteq Path(l)$ for every $l \in Lab$
- 2 since $\text{mvp}(S) = \text{mop}(S)$ in intraprocedural case, and by undecidability of MOP solution (cf. Theorem 7.4) □

- 1 Interprocedural Dataflow Analysis
- 2 Intraprocedural vs. Interprocedural Analysis
- 3 The MVP Solution
- 4 The Interprocedural Fixpoint Solution

- **Goal:** adapt fixpoint solution to **avoid invalid paths**
- **Approach:** encode call history into data flow properties (use **stacks** D^+ as dataflow version of runtime stack)
- Non-procedural constructs (**skip**, assignments, tests): operate only on **topmost element**
- **call:** computes **new topmost entry** from current and pushes it
- **return:** **removes topmost entry** and combines it with underlying (= call-site) entry

The Interprocedural Extension I

Definition 18.11 (Interprocedural extension (forward analysis))

Let $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system where $\varphi_{l_r} : D^2 \rightarrow D$ for each $(l_c, l_n, l_x, l_r) \in \text{iflow}$ (and $\varphi_l : D \rightarrow D$ otherwise).

The **interprocedural extension** of S is given by

$$\hat{S} := (Lab, E, F, (\hat{D}, \hat{\sqsubseteq}), \hat{\iota}, \hat{\varphi})$$

where

- $\hat{D} := D^+$
- $d_1 \dots d_n \hat{\sqsubseteq} d'_1 \dots d'_n$ iff $d_i \sqsubseteq d'_i$ for every $1 \leq i \leq n$
- $\hat{\iota} := \iota \in D^+$
- $\hat{\varphi}_l : D^+ \rightarrow D^+$ where
 - for each $l \in Lab \setminus \{l_c, l_r \mid (l_c, l_n, l_x, l_r) \in \text{iflow}\}$:

$$\hat{\varphi}_l(dw) := \varphi_l(d)w$$

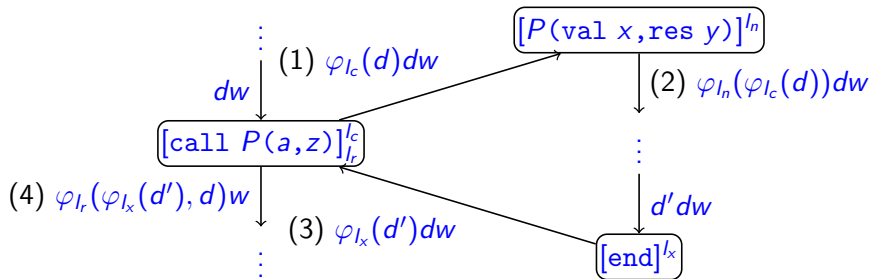
- for each $(l_c, l_n, l_x, l_r) \in \text{iflow}$:

$$\begin{aligned}\hat{\varphi}_{l_c}(dw) &:= \varphi_{l_c}(d)dw \\ \hat{\varphi}_{l_r}(d'dw) &:= \varphi_{l_r}(d', d)w\end{aligned}$$

The Interprocedural Extension II

Visualization of

- 1 $\hat{\varphi}_{l_c}(dw) := \varphi_{l_c}(d)dw$
- 2 $\hat{\varphi}_{l_n}(d'dw) := \varphi_{l_n}(d')dw$
- 3 $\hat{\varphi}_{l_x}(d'dw) := \varphi_{l_x}(d')dw$
- 4 $\hat{\varphi}_{l_r}(d'dw) := \varphi_{l_r}(d', d)w$



Example 18.12 (Constant Propagation (cf. Lecture 5/6))

$\hat{S} := (Lab, E, F, (\hat{D}, \hat{\sqsubseteq}), \hat{\iota}, \hat{\varphi})$ is determined by

- $D := \{\delta \mid \delta : Var_c \rightarrow \mathbb{Z} \cup \{\perp, \top\}\}$ (constant/undefined/overdefined)
- $\perp \sqsubseteq z \sqsubseteq \top$ for every $z \in \mathbb{Z}$
- $\iota := \delta_{\top} \in D$
- For each $l \in Lab \setminus \{l_c, l_n, l_x, l_r \mid (l_c, l_n, l_x, l_r) \in \text{iflow}\}$,

$$\varphi_l(\delta) := \begin{cases} \delta & \text{if } B^l = \text{skip or } B^l \in BExp \\ \delta[x \mapsto \text{val}_{\delta}(a)] & \text{if } B^l = (x := a) \end{cases}$$

- Whenever p_c contains $[\text{call } P(a, z)]_{l_r}^c$ and $\text{proc } [P(\text{val } x, \text{res } y)]_{l_n}^c \text{ is } c \text{ [end]}_{l_x}$,

- **call/entry:** set input/reset output parameter

$$\varphi_{l_c}(\delta) := \delta[x \mapsto \text{val}_{\delta}(a), y \mapsto \top], \quad \varphi_{l_n}(\delta) := \delta$$

- **exit/return:** reset parameters/set return value

$$\varphi_{l_x}(\delta) := \delta, \quad \varphi_{l_r}(\delta', \delta) := \delta'[x \mapsto \delta(x), y \mapsto \delta(y), z \mapsto \delta'(y)]$$