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 (~~Pro~~Seminar!)

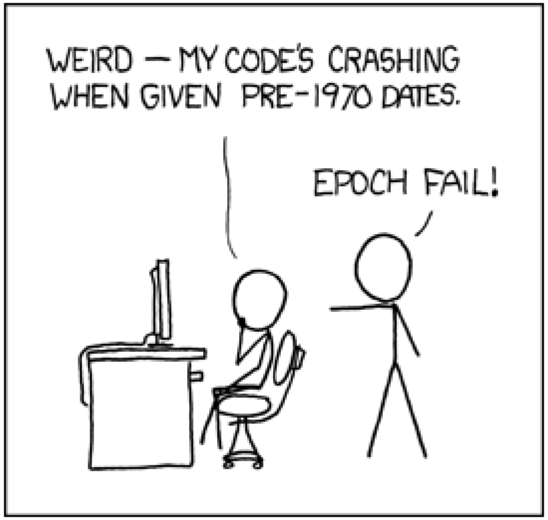
Where?

www.graphics.rwth-aachen.de/apse

When?

13.01.2017 – 29.01.2017

Seminar *Verification and Static Analysis of Software* (SS 2017)



https://xkcd.com/376

Topics

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

More information

`https://moves.rwth-aachen.de/teaching/ss-17/vsas/`

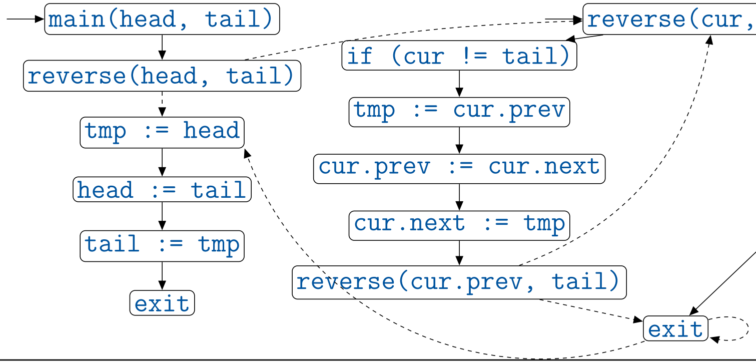
Registration

between January 13 and 29 via
`https://www.graphics.rwth-aachen.de/apse/`

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)



Extending the Syntax

Syntactic categories:

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

Context-free grammar:

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

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

An Example

Example 18.1 (Fibonacci numbers)

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

$$
\begin{array}{l}
\texttt{proc}\ [\texttt{Fib(val x, y, res z)}]^1\ \texttt{is} \\
\quad \texttt{if}\ [\texttt{x < 2}]^2\ \texttt{then} \\
\qquad [\texttt{z := y + 1}]^3 \\
\quad \texttt{else} \\
\qquad [\texttt{call Fib(x-1, y, z)}]^4_5; \\
\qquad [\texttt{call Fib(x-2, z, z)}]^6_7 \\
\quad \texttt{end} \\
[\texttt{end}]^8; \\
[\texttt{call Fib(5, 0, v)}]^9_{10}
\end{array}
$$

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:

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

Moreover the interprocedural flow of a program $p\,c$ is defined by

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

Procedure Flow Graphs II

Example 18.3 (Fibonacci numbers)

Flow graph of

$$
\begin{array}{l}
\texttt{proc}\ [\texttt{Fib(val x, y, res z)}]^1\ \texttt{is} \\
\quad \texttt{if}\ [\texttt{x < 2}]^2\ \texttt{then} \\
\qquad [\texttt{z := y + 1}]^3 \\
\quad \texttt{else} \\
\qquad [\texttt{call Fib(x-1, y, z)}]^4_5; \\
\qquad [\texttt{call Fib(x-2, z, z)}]^6_7 \\
\quad \texttt{end} \\
[\texttt{end}]^8; \\
[\texttt{call Fib(5, 0, v)}]^9_{10}
\end{array}
$$

(on the board)

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

Intraprocedural vs. Interprocedural Analysis

Naive Formulation I

- **Attempt:** directly transfer techniques from intraprocedural analysis
  $\Longrightarrow$ 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 $[\texttt{call}\ P(a,x)]^{l_c}_{l_r}$:
  transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \to D$ (definition later)
- For each procedure declaration $\texttt{proc}\ [P(\texttt{val}\ x, \texttt{res}\ y)]^{l_n}\ \texttt{is}\ c\ [\texttt{end}]^{l_x}$:
  transfer functions $\varphi_{l_n}, \varphi_{l_x} : D \to D$ (definition later)
- Induces equation system

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

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

Intraprocedural vs. Interprocedural Analysis

Naive Formulation II

Example 18.4 (Fibonacci numbers)

$$
\begin{array}{l}
\texttt{proc}\ [\texttt{Fib(val x, y, res z)}]^1\ \texttt{is} \\
\quad \texttt{if}\ [\texttt{x < 2}]^2\ \texttt{then} \\
\qquad [\texttt{z := y + 1}]^3 \\
\quad \texttt{else} \\
\qquad [\texttt{call Fib(x-1, y, z)}]^4_5; \\
\qquad [\texttt{call Fib(x-2, z, z)}]^6_7 \\
\quad \texttt{end} \\
[\texttt{end}]^8; \\
[\texttt{call Fib(5, 0, v)}]^9_{10}
\end{array}
$$

- "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)

```
proc [P(val x, res y)]^1 is
  [y := x]^2
[end]^3;
if [y = 0]^4 then
  [call P(1, y)]^5_6;
  [y := y - 1]^7
else
  [call P(2, y)]^8_9;
  [y := y - 2]^10
end;
[skip]^11
```

Two "valid" and two "invalid" paths:

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

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

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 $S = (Lab, E, F, (D, \sqsubseteq), \iota, \varphi)$ 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 grammar:

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

Valid Paths II

Example 18.7 (Fibonacci numbers; cf. Example 18.4)

$\texttt{proc}\ [\texttt{Fib(val x, y, res z)}]^1\ \texttt{is}$
$\quad\texttt{if}\ [\texttt{x < 2}]^2\ \texttt{then}$
$\qquad[\texttt{z := y + 1}]^3$
$\quad\texttt{else}$
$\qquad[\texttt{call Fib(x-1, y, z)}]_5^4;$
$\qquad[\texttt{call Fib(x-2, z, z)}]_7^6$
$\quad\texttt{end}$
$[\texttt{end}]^8;$
$[\texttt{call Fib(5, 0, v)}]_{10}^9$

Reminder:

$P[l_1, l_2] \to l_1$ for $l_1 = l_2$
$P[l_1, l_3] \to l_1, P[l_2, l_3]$ for $(l_1, l_2) \in F$
$P[l_c, l] \to l_c, P[l_n, l_x], P[l_r, l]$
for $(l_c, l_n, l_x, l_r) \in iflow$

Valid paths from 9 to 10:

$$
\begin{aligned}
P[9,10] &\to 9, P[1,8], P[10,10] \\
P[1,8] &\to 1, P[2,8] \\
P[2,8] &\to 2, P[3,8] \\
P[2,8] &\to 2, P[4,8] \\
P[3,8] &\to 3, P[8,8] \\
P[4,8] &\to 4, P[1,8], P[5,8] \\
P[5,8] &\to 5, P[6,8] \\
P[6,8] &\to 6, P[1,8], P[7,8] \\
P[7,8] &\to 7, P[8,8] \\
P[8,8] &\to 8 \\
P[10,10] &\to 10
\end{aligned}
$$

Thus $[9,1,2,3,8,10] \in L(P[9,10])$,
$[9,1,2,4,1,2,3,8,10] \notin L(P[9,10])$

The MVP Solution I

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

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

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

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

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

The MVP Solution II

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

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

where, for every $l \in Lab$,

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

Corollary 18.10

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

Proof.

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