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 (ProSeminar!)

Where?

www.graphics.rwth-aachen.de/apse

When?

13.01.2017 – 29.01.2017
Seminar Verification and Static Analysis of Software (SS 2017)

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/

https://xkcd.com/376
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)

```
main(head, tail)
reverse(head, tail)
tmp := head
head := tail
tail := tmp
exit
```

```
reverse(cur, tail)
if (cur != tail)
tmp := cur.prev
cur.prev := cur.next
cur.next := tmp
reverse(cur.prev, tail)
exit
if (cur = tail)
```

```
Interprocedural Dataflow Analysis

Extending the Syntax

Syntactic categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Domain</th>
<th>Meta variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure identifiers</td>
<td>Pid = {P, Q, \ldots}</td>
<td>P</td>
</tr>
<tr>
<td>Procedure declarations</td>
<td>PDec</td>
<td>p</td>
</tr>
<tr>
<td>Commands (statements)</td>
<td>Cmd</td>
<td>c</td>
</tr>
</tbody>
</table>

Context-free grammar:

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

- All labels and procedure names in program p c distinct
- In proc \([P(\text{val }x, \text{res } y)]^n \text{is } c \text{[end]}^x, l_n / l_x\) refers to the entry / exit of P
- In \([\text{call } P(a, x)]^l_c, l_c/l_r\) refers to the call of / return from P
- First parameter call-by-value (input), second call-by-result (output)
Example 18.1 (Fibonacci numbers)

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

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

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

The auxiliary functions \( \text{init} \), \( \text{final} \), and \( \text{flow} \) are extended as follows:

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

Moreover the interprocedural flow of a program \( p c \) is defined by

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

\[\subseteq \text{Lab}^4\]
Interprocedural Dataflow Analysis

Procedure Flow Graphs II

Example 18.3 (Fibonacci numbers)

Flow graph of

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

(on the board)

Here \( \text{iflow} = \{(9, 1, 8, 10), (4, 1, 8, 5), (6, 1, 8, 7)\} \)
Intraprocedural vs. Interprocedural Analysis

Naive Formulation I

- **Attempt:** directly transfer techniques from intraprocedural analysis
  \[ \Rightarrow \text{treat} (l_c; l_n) \text{ like} (l_c, l_n) \text{ and} (l_x; l_r) \text{ like} (l_x, l_r) \]
- **Given:** dataflow system \( S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \phi) \)
- For each procedure call \([\text{call } P(a, x)]^{l_c}\):
  transfer functions \( \phi_{l_c}, \phi_{l_r} : D \rightarrow D \) (definition later)
- For each procedure declaration \( \text{proc} \ [P(\text{val} \ x, \text{res} \ y)]^{l_n} \text{ is } c \ [\text{end}]^{l_x} \):
  transfer functions \( \phi_{l_n}, \phi_{l_x} : D \rightarrow D \) (definition later)
- **Induces equation system**
  \[
  AI_l = \begin{cases} 
  \iota & \text{if } l \in E \\
  \bigcup \{ \phi_{l'}(AI_{l'}) \mid (l', l) \in F \text{ or } (l'; l) \in F \} & \text{otherwise} 
  \end{cases}
  \]
- **Problem:** procedure calls \( (l_c; l_n) \) and procedure returns \( (l_x; l_r) \) treated like goto's
  \[ \Rightarrow \text{nesting of calls and returns ignored} \]
  \[ \Rightarrow \text{too many paths considered} \]
  \[ \Rightarrow \text{analysis information possibly imprecise} \text{ (but still correct)} \]
Intraprocedural vs. Interprocedural Analysis

Naive Formulation II

Example 18.4 (Fibonacci numbers)

\[
\text{proc} \left[ \text{Fib}(\text{val } x, y, \text{res } z) \right] \text{ is}
\]
\[
\text{if} \left[ x < 2 \right] \text{ then}
\]
\[
[z := y + 1]
\]
\[
\text{else}
\]
\[
[\text{call Fib}(x-1, y, z)]^4;
\]
\[
[\text{call Fib}(x-2, z, z)]^6
\]
\[
\text{end}
\]
\[
[\text{end}]^8;
\]
\[
[\text{call Fib}(5, 0, v)]^9
\]

- “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)] is
  [y := x];
[end];
if [y = 0] then
  [call P(1, y)];
  [y := y - 1];
else
  [call P(2, y)];
  [y := y - 2];
end;
[skip]

⇒ actually always y = 0 at 11, but naive method yields y = ⊤
```

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
The MVP Solution

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:

- $P[l_1, l_2] \rightarrow l_1$ whenever $l_1 = l_2$
- $P[l_1, l_3] \rightarrow l_1, P[l_2, l_3]$ whenever $(l_1, l_2) \in F$
- $P[l_c, l] \rightarrow l_c, P[l_n, l_x], P[l_r, l]$ whenever $(l_c, l_n, l_x, l_r) \in iflow$
Valid Paths II

Example 18.7 (Fibonacci numbers; cf. Example 18.4)

<table>
<thead>
<tr>
<th>proc Fib(val x, y, res z) is</th>
</tr>
</thead>
<tbody>
<tr>
<td>if [x &lt; 2] then [z := y + 1]</td>
</tr>
<tr>
<td>else [call Fib(x-1, y, z)];</td>
</tr>
<tr>
<td>[call Fib(x-2, z, z)]</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>[call Fib(5, 0, v)]</td>
</tr>
</tbody>
</table>

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 \([9, 1, 2, 3, 8, 10] \in L(P[9, 10]), [9, 1, 2, 4, 1, 2, 3, 8, 10] \not\in L(P[9, 10])\)
Definition 18.8 (Complete valid paths)

Let \( S = (\text{Lab}, E, F, (D, \sqsubseteq), \iota, \varphi) \) be a dataflow system. For every \( l \in \text{Lab} \), the set of valid paths up to \( l \) is given by

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

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

\[
\varphi_\pi := \varphi_{l_{k-1}} \circ \ldots \circ \varphi_{l_1} \circ \text{id}_D
\]

(so that \( \varphi[] = \text{id}_D \)).
The MVP Solution

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

\[
mvp(S) := (mvp(l_1), \ldots, mvp(l_n)) \in D^n
\]

where, for every \( l \in Lab \),

\[
mvp(l) := \bigcup \{\varphi_\pi(l) \mid \pi \in VPath(l)\}.
\]

Corollary 18.10

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

Proof.

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