Semantics and Verification of Software

Lecture 22: Dataflow Analysis IX (Interprocedural Fixpoint Solution)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/svsw08/

Winter semester 2008/09

Outline

Extending the Syntax

Syntactic categories:

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

Context-free grammar:

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

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

Procedure Flow Graphs

Definition (Procedure flow graphs)

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

$$\begin{aligned}
\mathsf{init}([\texttt{call } P(a,x)]^{l_c}_{l_r}) &:= l_c \\
\mathsf{final}([\texttt{call } P(a,x)]^{l_c}_{l_r}) &:= \{l_r\} \\
\mathsf{flow}([\texttt{call } P(a,x)]^{l_c}_{l_r}) &:= \{(l_c; l_n), (l_x; l_r)\}
\end{aligned}$$

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

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

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

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

Naive Formulation

Example (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_5;
    [call Fib(x-2, z, z)]^6_7;
[end]^8;
[call Fib(5, 0, v)]^9_10
```

- "Valid" path: $[9, 1, 2, 3, 8, 10]$
- "Invalid" path: $[9, 1, 2, 4, 1, 2, 3, 8, 10]$

Valid Paths I

- Consider only paths with correct nesting of procedure calls and returns
- Will yield MVP solution (Meet over all Valid Paths)

Definition (Valid paths I)

Given a dataflow system $S = (L, E, F, (D, \sqsubseteq), \iota, \varphi)$ and $l_1, l_2 \in L$, 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 productions:

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

The MVP Solution I

Definition (Valid paths II)

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

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

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

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

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

The MVP Solution II

Definition (MVP solution)

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

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

where, for every $l \in L$,

$$\mathsf{mvp}(l) := \bigsqcup\{\varphi_p(\iota) \mid p \in VPath(l)\}.$$

Corollary

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

Proof.

1. since $VPath(l) \subseteq Path(l)$ for every $l \in L$
2. by undecidability of MOP solution

□

Outline

Making Context Explicit

- **Goal:** adapt fixpoint solution to avoid invalid paths

Making Context Explicit

- **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)

Making Context Explicit

- **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

Making Context Explicit

- **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

Making Context Explicit

- **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 entry

The Interprocedural Extension I

Definition 22.1 (Interprocedural extension (forward analysis))

Let $S = (L, E, F, (D, \sqsubseteq), \iota, \varphi)$ be a dataflow system. The interprocedural extension of S is given by

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

where

- $\hat{D} := D^+$
- $d_1 \dots d_n \mathrel{\hat{\sqsubseteq}} d'_1 \dots d'_n$ iff $d_i \sqsubseteq d'_i$ for every $1 \leq i \leq n$
- $\hat{\iota} := \iota \in D^+$
- for each $l \in L \setminus \{l_c, l_n, l_x, l_r \mid (l_c, l_n, l_x, l_r) \in IF\}$, $\hat{\varphi}_l : D^+ \to D^+$ is given by $\hat{\varphi}_l(dw) := \varphi_l(d)w$
- for each $(l_c, l_n, l_x, l_r) \in IF$, $\hat{\varphi}_l : D^+ \to D^+$ is given by
  - $\hat{\varphi}_{l_c}(dw) := \varphi_{l_c}(d)dw$
  - $\hat{\varphi}_{l_n}(w) := w$
  - $\hat{\varphi}_{l_x}(w) := w$
  - $\hat{\varphi}_{l_r}(d'dw) := \varphi_{l_r}(d', d)w$

The Interprocedural Extension II

Remark: the schema

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

can be generalized by allowing a modification of the topmost entry in 2. and 3. (local variables, ...)

The Interprocedural Extension III

Example 22.2 (Constant Propagation (cf. Lecture 19))

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

- $D := \{\delta \mid \delta : Var_c \to \mathbb{Z} \cup \{\bot, \top\}\}$
- $\bot \sqsubseteq z \sqsubseteq \top$
- $\iota := \delta_\top \in D$
- for each $l \in L \setminus \{l_c, l_n, l_x, l_r \mid (l_c, l_n, l_x, l_r) \in IF\}$,
  $\varphi_l(\delta) := \begin{cases} \delta & \text{if } B^l = \texttt{skip} \text{ or } B^l \in BExp \\ \delta[x \mapsto \mathfrak{A}[\![a]\!]\delta] & \text{if } B^l = (x \texttt{ := } a) \end{cases}$
- whenever $p\,c$ contains $[\texttt{call } P(a, z)]^{l_c}_{l_r}$ and $\texttt{proc } [P(\texttt{val } x, \texttt{res } y)]^{l_n} \texttt{ is } c\ [\texttt{end}]^{l_x}$,
  - `call`: set input parameter and reset output parameter
    $\varphi_{l_c}(\delta) := \delta[x \mapsto \mathfrak{A}[\![a]\!]\delta, y \mapsto \top]$
  - return: propagate output parameter to caller by overwriting old value
    $\varphi_{l_r}(\delta', \delta) := \delta[z \mapsto \delta'(y)]$

Outline

The Equation System I

For an interprocedural dataflow system $\hat{S} := (L, E, F, (\hat{D}, \hat{\sqsubseteq}), \hat{\iota}, \hat{\varphi})$, the intraprocedural equation system

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

is extended to a system with three kinds of equations (for every $l \in L$):

- for actual dataflow information: $\mathsf{AI}_l \in D^+$ (extension of intraprocedural AI)
- for transfer functions of single nodes: $f_l : D^+ \to D^+$ (extension of intraprocedural transfer functions)
- for transfer functions of complete procedures: $F_l : D^+ \to D^+$ ($F_l(w)$ yields information at l if surrounding procedure is called with information $w \implies$ full procedure represented by F_{l_x})

The Equation System II

Formal definition:

$$\mathsf{AI}_l = \begin{cases} \iota & \text{if } l \in E \\ \bigsqcup\{\hat{\varphi}_{l_c}(\mathsf{AI}_{l_c}) \mid (l_c, l_n, l_x, l_r) \in IF\} & \text{if } l = l_n \\ & \text{for some } (l_c, l_n, l_x, l_r) \in IF \\ \bigsqcup\{f_{l'}(\mathsf{AI}_{l'}) \mid (l', l) \in F\} & \text{otherwise} \end{cases}$$

(if l not an exit label)

$$f_l(w) = \begin{cases} \hat{\varphi}_{l_r}(F_{l_x}(\hat{\varphi}_{l_c}(w))) & \text{if } l = l_c \text{ for some } (l_c, l_n, l_x, l_r) \in IF \\ \hat{\varphi}_l(w) & \text{otherwise} \end{cases}$$

$$F_l(w) = \begin{cases} w & \text{if } l = l_n \\ & \text{for some } (l_c, l_n, l_x, l_r) \in IF \\ \bigsqcup\{f_{l'}(F_{l'}(w)) \mid (l', l) \in F\} & \text{otherwise} \end{cases}$$

(if l occurs in procedure)

The Equation System II

Formal definition:

$$\mathsf{AI}_l = \begin{cases} \iota & \text{if } l \in E \\ \bigsqcup\{\hat{\varphi}_{l_c}(\mathsf{AI}_{l_c}) \mid (l_c, l_n, l_x, l_r) \in IF\} & \text{if } l = l_n \\ & \text{for some } (l_c, l_n, l_x, l_r) \in IF \\ \bigsqcup\{f_{l'}(\mathsf{AI}_{l'}) \mid (l', l) \in F\} & \text{otherwise} \end{cases}$$

(if l not an exit label)

$$f_l(w) = \begin{cases} \hat{\varphi}_{l_r}(F_{l_x}(\hat{\varphi}_{l_c}(w))) & \text{if } l = l_c \text{ for some } (l_c, l_n, l_x, l_r) \in IF \\ \hat{\varphi}_l(w) & \text{otherwise} \end{cases}$$

$$F_l(w) = \begin{cases} w & \text{if } l = l_n \\ & \text{for some } (l_c, l_n, l_x, l_r) \in IF \\ \bigsqcup\{f_{l'}(F_{l'}(w)) \mid (l', l) \in F\} & \text{otherwise} \end{cases}$$

(if l occurs in procedure)

As before: induces monotonic functional on lattice with ACC

$\Longrightarrow$ least fixpoint effectively computable

The Equation System III

Example 22.3 (Constant Propagation)

```
proc [P(val x, res y)]^1 is
  [y := 2*(x-1)]^2;
[end]^3;
[call P(2, z)]^4_5;
[call P(z, z)]^6_7;
[skip]^8
```