Static Program Analysis

Lecture 20: Extensions II
(Interprocedural Dataflow Analysis – 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/spa11/`

Summer Semester 2011

Outline

Extending the Syntax

Syntactic categories:

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

Context-free grammar:

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

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

Procedure Flow Graphs

Definition (Procedure flow graphs)

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

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

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

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

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

Naive Formulation

- **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 $S = (L, E, F, (D, \sqsubseteq), \iota, \varphi)$
- For each procedure call $[\texttt{call } P\texttt{(}a\texttt{,}x\texttt{)}]^{l_c}_{l_r}$:
  transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \to D$ (definition later)
- For each procedure declaration $\texttt{proc } [P\texttt{(val } x\texttt{,res } y\texttt{)}]^{l_n} \texttt{ is } c\ [\texttt{end}]^{l_x}$:
  transfer functions $\varphi_{l_n}, \varphi_{l_x} : D \to D$ (definition later)
- Induces equation system

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

- **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
  $\Longrightarrow$ analysis information imprecise (but still correct)

Valid Paths

- 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:

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

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

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

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

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

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

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

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

where, for every $l \in L$,

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

Corollary

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

Proof.

1. since $VPath(l) \subseteq Path(l)$ for every $l \in L$
2. since $\mathrm{mvp}(S) = \mathrm{mop}(S)$ in intraprocedural case, and by undecidability of MOP solution (cf. Theorem 6.2)

□

Outline

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 (= call-site) entry

The Interprocedural Extension I

Definition 20.1 (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

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

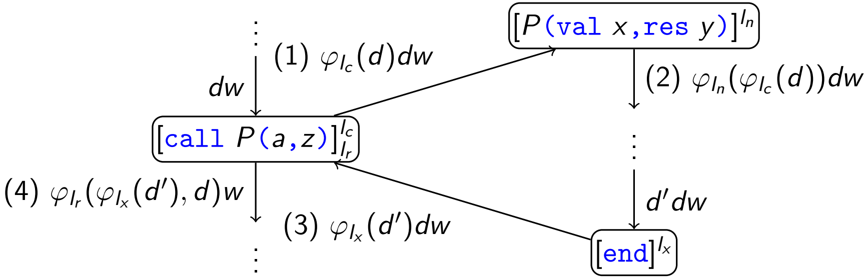
where

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

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. $\hat{\varphi}_{l_r}(d'dw) := \varphi_{l_r}(d', d)w$



dw
$[\texttt{call } P\texttt{(}a\texttt{,}z\texttt{)}]^{l_c}_{l_r}$
(1) $\varphi_{l_c}(d)dw$
$[P\texttt{(val } x\texttt{,res } y\texttt{)}]^{l_n}$
(2) $\varphi_{l_n}(\varphi_{l_c}(d))dw$
$d'dw$
$[\texttt{end}]^{l_x}$
(3) $\varphi_{l_x}(d')dw$
(4) $\varphi_{l_r}(\varphi_{l_x}(d'), d)w$

The Interprocedural Extension III

Example 20.2 (Constant Propagation (cf. Lecture 6))

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

- $D := \{\delta \mid \delta : Var_c \to \mathbb{Z} \cup \{\bot, \top\}\}$
- $\bot \sqsubseteq z \sqsubseteq \top$ for every $z \in \mathbb{Z}$
- $\iota := \delta_\top \in D$
- For each $l \in L \setminus \{l_c, l_n, l_x, l_r \mid (l_c, l_n, l_x, l_r) \in \text{iflow}\}$,
  $$\varphi_l(\delta) := \begin{cases} \delta & \text{if } B^l = \texttt{skip} \text{ or } B^l \in BExp \\ \delta[x \mapsto val_\delta(a)] & \text{if } B^l = (x \texttt{ := } a) \end{cases}$$
- Whenever $p\,c$ contains $[\texttt{call } P\texttt{(}a\texttt{,}z\texttt{)}]^{l_c}_{l_r}$ and $\texttt{proc } [P\texttt{(val } x\texttt{,res } y\texttt{)}]^{l_n} \texttt{ is } c\ [\texttt{end}]^{l_x}$,
  - call/entry: set input/reset output parameter
    $$\varphi_{l_c}(\delta) := \delta[x \mapsto val_\delta(a), y \mapsto \top], \quad \varphi_{l_n}(\delta) := \delta$$
  - exit/return: reset parameters/set return value
    $$\varphi_{l_x}(\delta) := \delta, \quad \varphi_{l_r}(\delta', \delta) := \delta'[x \mapsto \delta(x), y \mapsto \delta(y), z \mapsto \delta'(y)]$$

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 (cf. Definition 4.9)

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

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

- for actual dataflow information: $\mathsf{AI}_l \in D^+$ (counterpart of intraprocedural AI)
- for transfer functions of single nodes: $f_l : D^+ \to D^+$ (extension of intraprocedural transfer functions with special handling of procedure calls)
- 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 $w \implies$ complete procedure represented by F_{l_x})

The Equation System II

Formal definition – dataflow equations:

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

(if l not a return label)

Node transfer functions:

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

(if l not an exit or return label)

Procedure transfer functions:

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

(if l occurs in some procedure)

As before: induces monotonic functional on lattice with ACC

$\Longrightarrow$ least fixpoint effectively computable

The Equation System III

Example 20.3 (Constant Propagation)

Program:

```
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
```

Dataflow equations:

$$
\begin{aligned}
\mathrm{AI}_1 &= \hat{\varphi}_4(\mathrm{AI}_4) \sqcup \hat{\varphi}_6(\mathrm{AI}_6)\\
\mathrm{AI}_2 &= f_1(\mathrm{AI}_1)\\
\mathrm{AI}_3 &= f_2(\mathrm{AI}_2)\\
\mathrm{AI}_4 &= \iota = \top\top\top\\
\mathrm{AI}_6 &= f_4(\mathrm{AI}_4)\\
\mathrm{AI}_8 &= f_6(\mathrm{AI}_6)
\end{aligned}
$$

Node transfer functions:

$$
\begin{aligned}
\hat{\varphi}_1(\delta w) &= \delta w\\
\hat{\varphi}_2(\delta w) &= \delta[\mathtt{y} \mapsto val_\delta(\mathtt{2*(x-1)})]w\\
\hat{\varphi}_3(\delta w) &= \delta w\\
\hat{\varphi}_4(\delta w) &= \delta[\mathtt{x} \mapsto 2, \mathtt{y} \mapsto \top]\delta w\\
\hat{\varphi}_5(\delta'\delta w) &= \delta'[\mathtt{x} \mapsto \delta(\mathtt{x}), \mathtt{y} \mapsto \delta(\mathtt{y}), \mathtt{z} \mapsto \delta'(\mathtt{y})]w\\
\hat{\varphi}_6(\delta w) &= \delta[\mathtt{x} \mapsto \delta(\mathtt{z}), \mathtt{y} \mapsto \top]\delta w\\
\hat{\varphi}_7(\delta'\delta w) &= \delta'[\mathtt{x} \mapsto \delta(\mathtt{x}), \mathtt{y} \mapsto \delta(\mathtt{y}), \mathtt{z} \mapsto \delta'(\mathtt{y})]w
\end{aligned}
$$

$$
\begin{aligned}
f_1(\delta w) &= \hat{\varphi}_1(\delta w) = \delta w\\
f_2(\delta w) &= \hat{\varphi}_2(\delta w) = \delta[\mathtt{y} \mapsto val_\delta(\mathtt{2*(x-1)})]w\\
f_4(\delta w) &= \hat{\varphi}_5(\hat{\varphi}_3(F_3(\hat{\varphi}_4(\delta w)))) = \hat{\varphi}_5(F_3(\hat{\varphi}_4(\delta w)))\\
f_6(\delta w) &= \hat{\varphi}_7(\hat{\varphi}_3(F_3(\hat{\varphi}_6(\delta w)))) = \hat{\varphi}_7(F_3(\hat{\varphi}_6(\delta w)))\\
f_8(\delta w) &= \hat{\varphi}_8(\delta w) = \delta w
\end{aligned}
$$

Procedure transfer functions:

$$
\begin{aligned}
F_1(\delta w) &= \delta w\\
F_2(\delta w) &= f_1(F_1(\delta w)) = \delta w\\
F_3(\delta w) &= f_2(F_2(\delta w)) = \delta[\mathtt{y} \mapsto val_\delta(\mathtt{2*(x-1)})]w
\end{aligned}
$$

Fixpoint iteration:

on the board

The Fixpoint Iteration

For the fixpoint iteration it is important that the auxiliary functions only operate on the topmost element of the stack:

Lemma 20.4

For every $l \in L$, $d \in D$, and $w \in D^*$,

$$f_l(dw) = f_l(d)w \textit{ and } F_l(dw) = F_l(d)w$$

Proof.

see J. Knoop, B. Steffen: *The Interprocedural Coincidence Theorem*, Proc. CC '92, LNCS 641, Springer, 1992, 125–140 □

It therefore suffices to consider stacks with at most two entries, and so the fixpoint iteration ranges over "finitary objects".

Soundness and Completeness

The following results carry over from the intraprocedural case:

Theorem 20.5

Let $\hat{S} := (L, E, F, (\hat{D}, \hat{\sqsubseteq}), \hat{\iota}, \hat{\varphi})$ *be an interprocedural dataflow system.*

1. *(cf. Theorem 7.2)*
   $\text{mvp}(\hat{S}) \mathrel{\hat{\sqsubseteq}} \text{fix}(\Phi_{\hat{S}})$
2. *(cf. Theorem 7.5)*
   $\text{mvp}(\hat{S}) = \text{fix}(\Phi_{\hat{S}})$ *if all* $\hat{\varphi}_l$ *are distributive*

Proof.

see J. Knoop, B. Steffen: *The Interprocedural Coincidence Theorem*, Proc. CC '92, LNCS 641, Springer, 1992, 125–140 □

Outline

Context-Sensitive Interprocedural DFA

- **Observation:** MVP and fixpoint solution maintain proper relationship between procedure calls and returns
- **But:** do not distinguish between different procedure calls

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

  - information about calling states combined for all call sites
  - procedure body only analyzed once using combined information
  - resulting information used at all return points

$\Longrightarrow$ “context-insensitive”

- **Alternative:** context-sensitive analysis
  - separate information for different call sites
  - implementation by “procedure cloning”
  - more precise
  - more costly