

Semantics and Verification of Software

Lecture 21: Dataflow Analysis

Thomas Noll

Lehrstuhl für Informatik 2
RWTH Aachen University
noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/svsw/>

Summer semester 2007

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

- So far: only **intraprocedural analyses** (i.e., without user-defined functions or procedures)
- 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
 - mutually recursive procedures
 - one call-by-value and one call-by-result parameter
 - extension to multiple and call-by-value-result parameters straightforward

- So far: only **intraprocedural analyses** (i.e., without user-defined functions or procedures)
- 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
 - mutually recursive procedures
 - one call-by-value and one call-by-result parameter
 - extension to multiple and call-by-value-result parameters straightforward

- So far: only **intraprocedural analyses** (i.e., without user-defined functions or procedures)
- 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
 - mutually recursive procedures
 - one call-by-value and one call-by-result parameter
 - extension to multiple and call-by-value-result parameters straightforward

- So far: only **intraprocedural analyses** (i.e., without user-defined functions or procedures)
- 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
 - mutually recursive procedures
 - 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	$PVar = \{P, Q, \dots\}$	P
Procedure declarations	$PDec$	p
Commands (statements)	Cmd	c

Context-free grammar:

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

- All labels and procedure names in program pc 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
- Static scoping of procedures
- First parameter call-by-value, second call-by-result

Extending the Syntax

Syntactic categories:

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

Context-free grammar:

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

- 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
- Static scoping of procedures
- First parameter call-by-value, second call-by-result

Extending the Syntax

Syntactic categories:

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

Context-free grammar:

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

- All labels and procedure names in program pc 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
- Static scoping of procedures
- First parameter call-by-value, second call-by-result

(extension: multiple call-by-value parameters)

Example 21.1 (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)]45;
    [call Fib(x-2, z, z)]67;
  [end]8;
[call Fib(5, 0, v)]910
```

Definition 21.2 (Procedure flow graphs)

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

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

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

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

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

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

Example 21.3 (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)]45;
    [call Fib(x-2, z, z)]67;
  [end]8;
[call Fib(5, 0, v)]910
```

(on the board)

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

Example 21.3 (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)]45;
    [call Fib(x-2, z, z)]67;
  [end]8;
[call Fib(5, 0, v)]910
```

(on the board)

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

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

Naive Formulation I

- **Attempt:** directly transfer techniques from intraprocedural analysis

$\implies$ 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 $[\text{call } P(a, x)]_{l_r}^{l_c}$:

transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \rightarrow D$ (definition later)

- For each procedure declaration

$\text{proc } [P(\text{val } x, \text{res } y)]_{l_n}^{l_x} \text{ is } c \text{ [end]}^{l_x}$:

transfer functions $\varphi_{l_n}, \varphi_{l_x} : D \rightarrow D$ (definition later)

- Induces equation system

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

- **Problem:** procedure calls $(l_c; l_n)$ and procedure returns $(l_x; l_r)$ treated like goto's

$\implies$ nesting of calls and returns ignored

$\implies$ too many paths

$\implies$ analysis information imprecise (but still correct)

Naive Formulation I

- **Attempt:** directly transfer techniques from intraprocedural analysis

$\implies$ 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 $[\text{call } P(a, x)]_{l_r}^{l_c}$:

transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \rightarrow D$ (definition later)

- For each procedure declaration

$\text{proc } [P(\text{val } x, \text{res } y)]_{l_n}^{l_x} \text{ is } c \text{ [end]}^{l_x}$:

transfer functions $\varphi_{l_n}, \varphi_{l_x} : D \rightarrow D$ (definition later)

- Induces equation system

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

- **Problem:** procedure calls $(l_c; l_n)$ and procedure returns $(l_x; l_r)$ treated like goto's

$\implies$ nesting of calls and returns ignored

$\implies$ too many paths

$\implies$ analysis information imprecise (but still correct)

Naive Formulation I

- **Attempt:** directly transfer techniques from intraprocedural analysis

$\implies$ 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 $[\text{call } P(a, x)]_{l_r}^{l_c}$:

transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \rightarrow D$ (definition later)

- For each procedure declaration

$\text{proc } [P(\text{val } x, \text{res } y)]_{l_n}^{\text{is } c} [\text{end}]^{l_x}$:

transfer functions $\varphi_{l_n}, \varphi_{l_x} : D \rightarrow D$ (definition later)

- Induces **equation system**

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

- **Problem:** procedure calls $(l_c; l_n)$ and procedure returns $(l_x; l_r)$ treated like goto's

$\implies$ nesting of calls and returns ignored

$\implies$ too many paths

$\implies$ analysis information imprecise (but still correct)

Naive Formulation I

- **Attempt:** directly transfer techniques from intraprocedural analysis

$\implies$ 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 $[\text{call } P(a, x)]_{l_r}^{l_c}$:

transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \rightarrow D$ (definition later)

- For each procedure declaration

$\text{proc } [P(\text{val } x, \text{res } y)]_{l_n}^{\text{is } c} [\text{end}]^{l_x}$:

transfer functions $\varphi_{l_n}, \varphi_{l_x} : D \rightarrow D$ (definition later)

- Induces **equation system**

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

- **Problem:** procedure calls $(l_c; l_n)$ and procedure returns $(l_x; l_r)$ treated like goto's

$\implies$ nesting of calls and returns ignored

$\implies$ too many paths

$\implies$ analysis information imprecise (but still correct)

Naive Formulation I

- **Attempt:** directly transfer techniques from intraprocedural analysis

$\implies$ 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 $[\text{call } P(a, x)]_{l_r}^{l_c}$:

transfer functions $\varphi_{l_c}, \varphi_{l_r} : D \rightarrow D$ (definition later)

- For each procedure declaration

$\text{proc } [P(\text{val } x, \text{res } y)]^{l_n} \text{ is } c \text{ [end]}^{l_x}$:

transfer functions $\varphi_{l_n}, \varphi_{l_x} : D \rightarrow D$ (definition later)

- Induces **equation system**

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

- **Problem:** procedure calls $(l_c; l_n)$ and procedure returns $(l_x; l_r)$ treated like goto's

$\implies$ nesting of calls and returns ignored

$\implies$ too many paths

$\implies$ analysis information imprecise (but still correct)

Example 21.4 (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 21.4 (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)]45;
    [call Fib(x-2, z, z)]67;
  [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]

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

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

Definition 21.5 (Valid paths I)

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 productions:

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

Example 21.6 (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)]54;
    [call Fib(x-2, z, z)]76;
  [end]8;
[call Fib(5, 0, v)]109
```

Thus

$[9, 1, 2, 3, 8, 10] \in L(P[10, 11])$

but

$[9, 1, 2, 4, 1, 2, 3, 8, 10] \notin L(P[10, 11])$

Grammar for $P[10, 11]$:
on the board

Example 21.6 (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)]₅⁴;

 [call Fib(x-2, z, z)]₇⁶;

 [end]⁸;

 [call Fib(5, 0, v)]₁₀⁹

Thus

$[9, 1, 2, 3, 8, 10] \in L(P[10, 11])$

but

$[9, 1, 2, 4, 1, 2, 3, 8, 10] \notin L(P[10, 11])$

Grammar for $P[10, 11]$:

on the board

Definition 21.7 (Valid paths II)

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

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

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

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

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

Definition 21.8 (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 $l \in Lab$,

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

Corollary 21.9

- 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 by undecidability of MOP solution



Definition 21.8 (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 $l \in Lab$,

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

Corollary 21.9

- 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 by undecidability of MOP solution



Definition 21.8 (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 $l \in Lab$,

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

Corollary 21.9

- 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 by undecidability of MOP solution

