Class 6

- Review; questions
- Assign (see Schedule for links)
  - Slicing overview (cont'd)
  - Problem Set 3: due 9/8/09

1

Program Slicing

2

Program Slicing

1. Slicing overview
2. Types of slices, levels of slices
3. Methods for computing slices
4. Interprocedural slicing (later)

3

Slicing Overview

Types of slices

- Backward static slice
- Executable slice
- Forward static slice
- Dynamic slice
- Execution slice
- Generic algorithm for static slice

Levels of slices

- Intraprocedural
- Interprocedural

Authors of articles

- Program Slicing
- A Survey of Program Slicing Techniques

1. Agrawal
2. Binkley
3. Gallagher
4. Gupta
5. Horgan
6. Horwitz
7. Korel
8. Laski
9. K. Ottenstein
10. L. Ottenstein
11. Reps
12. Soffa
13. Tip
14. Weiser

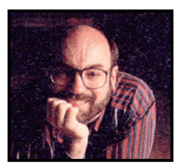
4

Some History

- Who first defined slicing?
- Why?

Some History

1. Mark Weiser, 1981



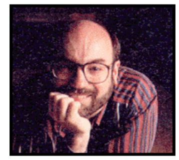
Experimented with programmers to show that slices are:

"The mental abstraction people make when they are debugging a program" [Weiser]

Used Data Flow Equations

Some History

1. Mark Weiser, 1981



   Experimented with programmers to show that slices are:

   "The mental abstraction people make when they are debugging a program" [Weiser]

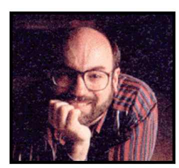
   Used Data Flow Equations

2. Ottenstein & Ottenstein – PDG, 1984

7

Some History

1. Mark Weiser, 1981

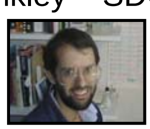


   Experimented with programmers to show that slices are:

   "The mental abstraction people make when they are debugging a program" [Weiser]

   Used Data Flow Equations

2. Ottenstein & Ottenstein – PDG, 1984
3. Horowitz, Reps & Binkley – SDG, 1990



8

Applications

- Debugging
- Program Comprehension
- Reverse Engineering
- Program Testing
- Measuring Program—metrics
  - Coverage, Overlap, Clustering
- Refactoring

9

Static VS Dynamic Slicing

- Static Slicing
  - Statically available information only
  - No assumptions made on input
  - Computed slice is in general inaccurate
  - Identifying minimal slices is an undecidable problem → approximations
  - Results may not be useful
- Dynamic Slicing
  - Computed on a given input
  - *Actual* instead of *may*
  - Useful for applications such as debugging and testing

11

Example

```
1.  read (n)
2.  i := 1
3.  sum := 0
4.  product := 1
5.  while i <= n do
6.      sum := sum + i
7.      product := product * i
8.      i := i + 1
9.  write (sum)
10. write (product)
```

Create CFG for program to use with subsequent examples

Types of Slicing (Backward Static)

```
1.  read (n)
2.  i := 1
3.  sum := 0
4.  product := 1
5.  while i <= n do
6.      sum := sum + i
7.      product := product * i
8.      i := i + 1
9.  write (sum)
10. write (product)
```

With respect to statement 10 and variable product

Types of Slicing (Backward Static)

A backward slice of a program with respect to a program point **p** and set of program variables **V**

Types of Slicing (Backward Static)

A backward slice of a program with respect to a program point **p** and set of program variables **V** consists of all statements and predicates in the program that may affect the value of variables in **V** at **p**

The program point **p** and the set of variables **V** together form the slicing criterion, usually written <**p, V**>

Types of Slicing (Backward Static)

General approach: backward traversal of program flow

- Slicing starts from point *p* (C = (*p* , V))
- Examines statements that could be executed before *p* (not just statements that appear before *p*)
- Add statements that affect value of V at *p* or execution to get to *p*
- Considers transitive dependencies

16

Types of Slicing (Backward Static)

1. read (n)
2. i := 1
3. sum := 0
4. product := 1
5. while i <= n do
6.   sum := sum + i
7.   product := product * i
8.   i := i + 1
9. write (sum)
10. write (product)

Criterion <10, product>

What is the backward slice?

Types of Slicing (Backward Static)

```
1.  read (n)
2.  i := 1
3.  sum := 0
4.  product := 1
5.  while i <= n do
6.      sum := sum + i
7.      product := product * i
8.      i := i + 1
9.  write (sum)
10. write (product)
```

Criterion <10, product>

Types of Slicing (Executable)

A slice is executable if the statements in the slice form a syntactically correct program that can be executed.

If the slice is computed correctly (safely), the result of running the program that is the executable slice produces the same result for variables in **V** at **p** for all inputs.

Types of Slicing (Executable)

```
1.  read (n)
2.  i := 1
3.  sum := 0
4.  product := 1
5.  while i <= n do
6.      sum := sum + i
7.      product := product * i
8.      i := i + 1
9.  write (sum)
10. write (product)
```

Is this slice executable?

Criterion <10, product>

```
1.  read (n)
2.  i := 1
3.
4.  product := 1
5.  while i <= n do
6.
7.      product := product * i
8.      i := i + 1
9.
10. write (product)
```

Types of Slicing (Forward Static)

A forward slice of a program with respect to a program point **p** and set of program variables **V** consists of all statements and predicates in the program that may be affected the value of variables in **V** at **p**

The program point **p** and the variables **V** together form the slicing criterion, usually written <**p, V**>

Types of Slicing (Forward Static)

```
1.  read (n)
2.  i := 1
3.  sum := 0
4.  product := 1
5.  while i <= n do
6.      sum := sum + i
7.      product := product * i
8.      i := i + 1
9.  write (sum)
10. write (product)
```

Criterion <3, sum>

What is the forward slice?

Types of Slicing (Forward Static)

```
1.  read (n)
2.  i := 1
3.  sum := 0
4.  product := 1
5.  while i <= n do
6.      sum := sum + i
7.      product := product * i
8.      i := i + 1
9.  write (sum)
10. write (product)
```

Criterion <3, sum>

Types of Slicing (Forward Static)

```
1.  read (n)
2.  i := 1
3.  sum := 0
4.  product := 1
5.  while i <= n do
6.      sum := sum + i
7.      product := product * i
8.      i := i + 1
9.  write (sum)
10. write (product)
```

Criterion <1, n>

What is the forward slice?

Types of Slicing (Forward Static)

```
1.  read (n)
2.  i := 1
3.  sum := 0
4.  product := 1
5.  while i <= n do
6.      sum := sum + i
7.      product := product * i
8.      i := i + 1
9.  write (sum)
10. write (product)
```

Criterion <1, n>

Types of Slicing (Dynamic)

A dynamic slice of a program with respect to an input value of a variable **v** at a program point **p** for a particular execution **e** of the program is the set of all statements in the program that affect the value of **v** at **p.**

The program point **p,** the variables **V,** and the input i for **e** form the slicing criterion, usually written <i**, v, p**>. The slicing uses the execution history or trajectory for the program with input **i.**

Types of Slicing (Dynamic)

A dynamic slice of a program with respect to an input value of a variable **v** at a program point **p** for a particular execution **e** of the program is

Types of Slicing (Dynamic)

A dynamic slice of a program with respect to an input value of a variable **v** at a program point **p** for a particular execution **e** of the program is the set of all statements in the program that affect the value of **v** at **p** during execution **e.**

The program point **p,** the variables **V,** and the input i for **e** form the slicing criterion, usually written <i**, v, p**>. The slicing uses the execution history or trajectory for the program with input **i.**

Types of Slicing (Dynamic)

```
read (n)
for I := 1 to n do
  a := 2
   if c1 then
      if c2 then
         a := 4
      else
         a := 6
      z := a
write (z)
```

Types of Slicing (Dynamic)

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.      if c1 then
5.         if c2 then
6.             a := 4
7.         else
8.             a := 6
9.         z := a
10.  write (z)
```

Input n is 1; c1, c2 both true

Execution history is

$1^1, 2^1, 3^1, 4^1, 5^1, 6^1, 9^1,$
$2^2, 10^1$

Criterion<1, 10^1, z>

What is the dynamic slice?

Types of Slicing (Dynamic)

```
1.   read (n)
2.   for I := 1 to n do (1)
3.     a := 2
4.      if c1 then
5.         if c2 then
6.             a := 4
9.      z := a
2.   for I := 1 to n do (2)
10.  write (z)
```

Input n is 1; c1, c2 both true

Execution history is

$1^1, 2^1, 3^1, 4^1, 5^1, 6^1, 9^1,$
$2^2, 10^1$

Criterion<1, 10^1, z>

What is the dynamic slice?

Types of Slicing (Dynamic)

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.          if c2 then
6.              a := 4
7.          else
8.              a := 6
9.      z := a
10.  write (z)
```

Input n is 1; c1, c2 both true

Execution history is

1^1, 2^1, 3^1, 4^1, 5^1, 6^1, 9^1, 2^2, 10^1

Criterion<1, 10^1, z>

Comparison of Static and Dynamic

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.          if c2 then
6.              a := 4
7.          else
8.              a := 6
9.      z := a
10.  write (z)
```

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.          if c2 then
6.              a := 4
7.          else
8.              a := 6
9.          z := a
10.  write (z)
```

What is the static slice for <10,z>?

Comparison of Static and Dynamic

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.      z := a
10.  write (z)
```

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.         z := a
10.  write (z)
```

Static slice <10, z>

Types of Slicing (Dynamic)

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.      if c1 then
5.         if c2 then
6.            a := 4
7.         else
8.            a := 6
9.      z := a
10.  write (z)
```

Input n is 2; c1, c2 false on first iteration and true on second iteration

Execution history is

$1^1, 2^1, 3^1, 4^1, 9^1, 2^2, 3^2, 4^2, 5^1, 6^1, 9^2, 2^3, 10^1$>

Criterion<1, 10^1, z>

What is the dynamic slice?

Types of Slicing (Dynamic)

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.       if c1 then
5.           if c2 then
6.               a := 4
7.           else
8.               a := 6
9.       z := a
10.  write (z)
```

Input n is 2; c1, c2 false on first iteration and true on second iteration

Execution history is

1^1, 2^1, 3^1, 4^1, 9^1, 2^2, 3^2, 4^2, 5^1, 6^1, 9^2, 2^3, 10^1>

Criterion<1, 10^1, z>

Types of Slicing (Dynamic)

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.       if c1 then
5.           if c2 then
6.               a := 4
7.           else
8.               a := 6
9.       z := a
10.  write (z)
```

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.       if c1 then
5.           if c2 then
6.               a := 4
7.           else
8.               a := 6
9.           z := a
10.  write (z)
```

Static slice <10, z>

Types of Slicing (Execution)

An execution slice of a program with respect to an input value of a variable **v** is the set of statements in the program that are executed with input **v.**

Types of Slicing (Execution)

```
1.   read (n)
2.   for I := 1 to n do
3.     a := 2
4.      if c1 then
5.          if c2 then
6.              a := 4
7.          else
8.              a := 6
9.      z := a
10.  write (z)
```

Input n is 2; c1, c2 false on first iteration and true on second iteration

Execution history is

$1^1, 2^1, 3^1, 4^1, 9^1, 2^2, 3^2, 4^2, 5^1, 6^1, 9^2, 2^3, 10^1$>

What is the execution slice?

Types of Slicing (Execution)

```
1.   read (n)
2.   for I := 1 to n do
3.      a := 2
4.       if c1 then
5.           if c2 then
6.               a := 4
7.           else
8.               a := 6
9.       z := a
10.  write (z)
```

Input n is 2; c1, c2 false on first iteration and true on second iteration

Execution history is

$1^1, 2^1, 3^1, 4^1, 9^1, 2^2, 3^2, 4^2, 5^1, 6^1, 9^2, 2^3, 10^1$>

Execution slice is

1, 2, 3, 4, 5, 6, 9, 10

Recap of Types of Slicing

- Static backward
- Executable
- Static forward
- Dynamic
- Execution

41

Methods for Computing Slices

- Data-flow on the flow graph
  - Intraprocedural: control-flow graph (CFG)
  - Interprocedural: interprocedural control-flow graph (ICFG) (later)
- Reachability in a dependence graph
  - Intraprocedural: program-dependence graph (PDG)
  - Interprocedural: system-dependence graph (SDG) (later)

Methods (Data-Flow on the CFG)

Data Flow Equations (Weiser)

- Iterative process (over CFG)
  - Compute consecutive sets of “relevant” variables for each node in the CFG using data dependencies
  - Control dependences are not computed explicitly
  - Variables of control predicates (if, while) are “indirectly relevant” if any one of the statements in their body is relevant
- Start with slicing criterion: C = (p, V)
- Continue until a fixed point is reached (i.e., last iteration does not find new relevant statements)

Methods (Data Flow on the CFG)

Definitions

- $i \longrightarrow_{CFG} j$: there is a directed edge from i to j
- Def(i): set of variables modified at statement i
- Ref(i): variables referenced at statement I
- Infl(i): Set of nodes that are influenced by i (control dependent)
- R^0_C: Directly relevant variables
- R^k_C: Indirectly relevant variables
- S^0_C: Directly relevant statements
- S^k_C: Indirectly relevant statements
- B^k_C: Relevant branch statements

44

Methods (Data Flow on the CFG)

Definitions

- $i \longrightarrow_{CFG} j$: there is a directed edge from i to j
- Def(i): set of variables modified at statement i
- Ref(i): variables referenced at statement I

  Local

- Infl(i): Set of nodes that are influenced by i (control dependent)

  Computed using CFG

- R^0_C: Directly relevant variables
- R^k_C: Indirectly relevant variables
- S^0_C: Directly relevant statements
- S^k_C: Indirectly relevant statements
- B^k_C: Relevant branch statements

  Propagated using CFG

45

Methods (Data Flow on the CFG)

Criterion <10, product>

```
1.  read (n)
2.  i := 1
3.  sum := 0
4.  product := 1
5.  while i <= n do
6.      sum := sum + i
7.      product := product * i
8.      i := i + 1
9.  write (sum)
10. write (product)
```

Methods (Data Flow on the CFG)

Node	*Def*	*Ref*	*Infl*	R_C^0	in S_C^0	in B_C^0	R_C^1	in S_C^1
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

Methods (Data Flow on the CFG)

Node	*Def*	*Ref*	*Infl*	R_C^0	in S_C^0	in B_C^0	R_C^1	in S_C^1
1	$\{n\}$	$\varnothing$	$\varnothing$					
2	$\{i\}$	$\varnothing$	$\varnothing$					
3	$\{sum\}$	$\varnothing$	$\varnothing$					
4	$\{product\}$	$\varnothing$	$\varnothing$					
5	$\varnothing$	$\{i, n\}$	$\{6, 7, 8\}$					
6	$\{sum\}$	$\{sum, i\}$	$\varnothing$					
7	$\{product\}$	$\{product, i\}$	$\varnothing$					
8	$\{i\}$	$\{i\}$	$\varnothing$					
9	$\varnothing$	$\{sum\}$	$\varnothing$					
10	$\varnothing$	$\{product\}$	$\varnothing$					

48

Methods (Data Flow on the CFG)

Iteration 0:

$$R_C^0(i) = V \text{ when } i = p$$

Variables used at criterion point are added; looking for definitions that affect these uses

$R_C^0(i)$ (for every i —> j)

(i) $v \in Ref(i)$ and $Def(i) \cap R_C^0(j) \neq \varnothing$

(ii), $v \notin Def(i)$ and $v \in R_C^0(j)$

$$S_C^0 = \{i \mid Def(i) \cap R_C^0(j) \neq \varnothing, i \rightarrow_{CFG} j\}$$

$$B_C^0 = \{b \mid i \in Infl(b), i \in S_C^0\}$$

49

Methods (Data Flow on the CFG)

Iteration 0:

$R_C^0(i) = V$ when $i = p$

$R_C^0(i)$ (for every i —> j) CFG predecessors

(i) $v \in Ref(i)$ and $Def(i) \cap R_C^0(j) \neq \emptyset$

(ii), $v \notin Def(i)$ and $v \in R_C^0(j)$

$S_C^0 = \{i \mid Def(i) \cap R_C^0(j) \neq \emptyset, i \rightarrow_{CFG} j\}$

$B_C^0 = \{b \mid i \in Infl(b), i \in S_C^0\}$

50

Methods (Data Flow on the CFG)

Iteration 0:

$R_C^0(i) = V$ when $i = p$

$R_C^0(i)$ (for every i —> j) CFG predecessors

(i) $v \in Ref(i)$ and $Def(i) \cap R_C^0(j) \neq \emptyset$

If variables in Ref(successor) defined in I (killed), add variables used in i to Ref(i) (new variables to consider)

(ii), $v \notin Def(i)$ and $v \in R_C^0(j)$

$S_C^0 = \{i \mid Def(i) \cap R_C^0(j) \neq \emptyset, i \rightarrow_{CFG} j\}$

$B_C^0 = \{b \mid i \in Infl(b), i \in S_C^0\}$

51

Methods (Data Flow on the CFG)

Iteration 0:

$R_C^0(i) = V$ when $i = p$

$R_C^0(i)$ (for every i —> j) CFG predecessors

(i) $v \in Ref(i)$ and $Def(i) \cap R_C^0(j) \neq \varnothing$

If variables in Ref(successor) defined in I (killed), add variables used in i to Ref(i) (new variables to consider)

(ii), $v \notin Def(i)$ and $v \in R_C^0(j)$

If variables in Ref(successor) and not defined in i, add variables in Ref(successor) to Ref(i) (propagate)

$$S_C^0 = \{i \mid Def(i) \cap R_C^0(j) \neq \varnothing, i \rightarrow_{CFG} j\}$$

$$B_C^0 = \{b \mid i \in Infl(b), i \in S_C^0\}$$

52

Methods (Data Flow on the CFG)

Iteration 0:

$R_C^0(i) = V$ when $i = p$

$R_C^0(i)$ (for every i —> j)

(i) $v \in Ref(i)$ and $Def(i) \cap R_C^0(j) \neq \varnothing$

(ii), $v \notin Def(i)$ and $v \in R_C^0(j)$

$$S_C^0 = \{i \mid Def(i) \cap R_C^0(j) \neq \varnothing, i \rightarrow_{CFG} j\}$$

If i defined a variable in R of successor, then add to S (this is the slice) because it has influence on variable

$$B_C^0 = \{b \mid i \in Infl(b), i \in S_C^0\}$$

53

Methods (Data Flow on the CFG)

Iteration 0:

$R_C^0(i) = V$ when $i = p$

$R_C^0(i)$ (for every i —> j)

(i) $v \in Ref(i)$ and $Def(i) \cap R_C^0(j) \neq \emptyset$

(ii), $v \notin Def(i)$ and $v \in R_C^0(j)$

$S_C^0 = \{i \mid Def(i) \cap R_C^0(j) \neq \emptyset, i \rightarrow_{CFG} j\}$

$B_C^0 = \{b \mid i \in Infl(b), i \in S_C^0\}$

If i has an influence (control dependence), it is in this set.

54

Methods (Data Flow on the CFG)

Iteration 0:

$R_C^0(i) = V$ when $i = p$

$R_C^0(i)$ (for every i $\longrightarrow_{CFG}$ j)

(i) $v \in Ref(i)$ and $Def(i) \cap R_C^0(j) \neq \emptyset$

(ii), $v \notin Def(i)$ and $v \in R_C^0(j)$

$S_C^0 = \{i \mid Def(i) \cap R_C^0(j) \neq \emptyset, i \rightarrow_{CFG} j\}$

$B_C^0 = \{b \mid i \in Infl(b), i \in S_C^0\}$

Iteration k+1:

$$R_C^{k+1}(i) = R_C^k(i) \cup \bigcup_{b \in B_C^k} R^0_{(b, Ref(b))}(i)$$

$$S_C^{k+1} = \{i \mid Def(i) \cap R_C^{k+1}(j) \neq \emptyset, i \rightarrow_{CFG} j\} \cup B_C^k$$

$$B_C^{k+1} = \{b \mid i \in Infl(b), i \in S_C^{k+1}\}$$

55

Methods (Data Flow on the CFG)

Node	*Def*	*Ref*	*Infl*	R_C^0	in S_C^0	in B_C^0	R_C^1	in S_C^1
1	$\{n\}$	$\emptyset$	$\emptyset$					
2	$\{i\}$	$\emptyset$	$\emptyset$					
3	$\{sum\}$	$\emptyset$	$\emptyset$					
4	$\{product\}$	$\emptyset$	$\emptyset$					
5	$\emptyset$	$\{i, n\}$	$\{6, 7, 8\}$					
6	$\{sum\}$	$\{sum, i\}$	$\emptyset$					
7	$\{product\}$	$\{product, i\}$	$\emptyset$					
8	$\{i\}$	$\{i\}$	$\emptyset$					
9	$\emptyset$	$\{sum\}$	$\emptyset$					
10	$\emptyset$	$\{product\}$	$\emptyset$					

56

Methods (Data Flow on the CFG)

Node	*Def*	*Ref*	*Infl*	R_C^0	in S_C^0	in B_C^0	R_C^1	in S_C^1
1	$\{n\}$	$\emptyset$	$\emptyset$	$\emptyset$			$\emptyset$	√
2	$\{i\}$	$\emptyset$	$\emptyset$	$\emptyset$	√		$\{n\}$	√
3	$\{sum\}$	$\emptyset$	$\emptyset$	$\{i\}$			$\{i, n\}$	
4	$\{product\}$	$\emptyset$	$\emptyset$	$\{i\}$	√		$\{i, n\}$	√
5	$\emptyset$	$\{i, n\}$	$\{6, 7, 8\}$	$\{product, i\}$		√	$\{product, i, n\}$	√
6	$\{sum\}$	$\{sum, i\}$	$\emptyset$	$\{product, i\}$			$\{product, i, n\}$	
7	$\{product\}$	$\{product, i\}$	$\emptyset$	$\{product, i\}$	√		$\{product, i, n\}$	√
8	$\{i\}$	$\{i\}$	$\emptyset$	$\{product, i\}$	√		$\{product, i, n\}$	√
9	$\emptyset$	$\{sum\}$	$\emptyset$	$\{product\}$			$\{product\}$	
10	$\emptyset$	$\{product\}$	$\emptyset$	$\{product\}$			$\{product\}$	

Note: You may not get these results for first iteration if propagation is done in a different order.

57