Program Slicing

Gias Uddin

Special Topic Lecture
prepared from the Survey of Frank Tip on Program Slicing:
Frank Tip. A Survey of Program Slicing Techniques. Technical Report, CWI (Centre for Mathematics and Computer Science) Amsterdam, The Netherlands, 1994. 64 pages.

Definition

Program slicing: is the task of computing program slices.

Program slice: consists of the parts of the program that (potentially) affect the values computed at some point of interest.

Slicing criterion: is a point of interest for slicing. It is typically specified by a location in the program in combination with a subset of the program's variables.

The parts of a program that have a direct or indirect effect on the values computed at a slicing criterion C constitute the *program slice with respect to the criterion* C.

Application of Program Slicing

- Debugging and program analysis: pinpoint a buggy statement and slice the program to find all the relevant statements and variables that may have been responsible for the buggy statement. (backward)
- Debugging and program analysis: pinpoint a buggy statement and slice the program to find all the relevant statements and variables to ensure that they are also not buggy. (forward)
- Program Differencing: analyze the old and new versions of the same program to determine how the new program components in the new version affect the old version.
- Software Maintenance: check whether the change at one point of a program may affect the behavior of the other parts of the program.
- Testing: a program satisfies a "conventional" dataflow testing criterion if all the def-use pairs occur in successful test-cases.
- Tuning Compilers: check potential occurrences of redundant sub-expressions. A common sub-expression occurs in all execution paths if its inputs are the same in all executions.

Types of Program Slicing

Two types: Static and Dynamic.

Static is computed without making assumptions regarding a program's inputs.

Dynamic relies on some specific test case.

Features of programming languages such as *procedures, unstructured control flow, composite data types and pointers, and concurrency* each require specific extensions of slicing algorithms.

Static Slicing (Overview Example)

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

(a)

```
read(n);
i := 1;

product := 1;
while i <= n do
begin

  product := product * i;
  i := i + 1
end;

write(product)
```

(b)

Figure 1: **(a)** An example program. **(b)** A slice of the program w.r.t. criterion (10, product).

All computations not relevant to the (final value of) variable product at line 10 have been "sliced away".

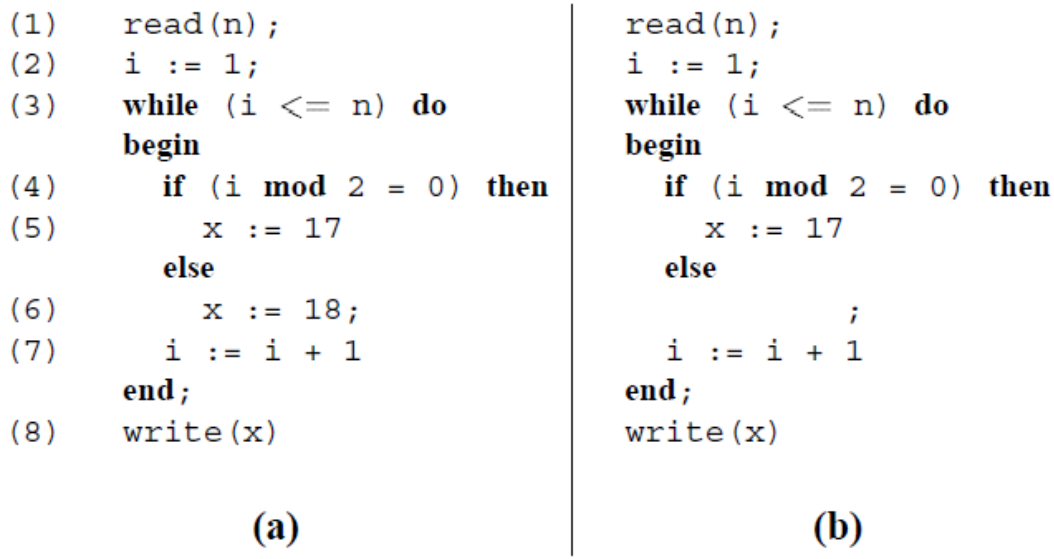
Ways of computing static slicing

Two ways: 1) backward traversal, 2) forward traversal.

Backward traversal: The example shown in the previous slide is an example of backward traversal. It is computed by gathering statements and control predicates by way of a backward traversal of the program's control flow graph (CFG) or program dependence graph (PDG), starting at the slicing criterion. These slices are called backward (static) slices.

Forward traversal: A forward slice consists of all the statements and control predicates dependent on the slicing criterion, a statement being "dependent" on the slicing criterion if the values computed at the statement depend on the values computed at the slicing criterion, or if the values computed at the slicing criterion determine the fact if the statement under consideration is executed or not.

Dynamic Slicing (Overview Example)



```
(1)   read(n);
(2)   i := 1;
(3)   while (i <= n) do
      begin
(4)     if (i mod 2 = 0) then
(5)       x := 17
        else
(6)       x := 18;
(7)     i := i + 1
      end;
(8)   write(x)
```

(a)

```
read(n);
i := 1;
while (i <= n) do
begin
  if (i mod 2 = 0) then
    x := 17
  else
        ;
  i := i + 1
end;
write(x)
```

(b)

Figure 2: **(a)** Another example program. **(b)** Dynamic slice w.r.t. criterion (n = 2, 8^1, x).

8^1 denotes the first occurrence of statement at line 8 in the execution history of the program.

The **else** branch of the **if** statement may be omitted from the dynamic slice since the assignment of 18 to variable x in the first iteration of the loop is "killed" by the assignment of 17 to x in the second iteration.

Dynamic Slicing Basics

Only the dependencies that occur in a *specific* execution of the program are taken into account.

A *dynamic slicing criterion* specifies the input, and distinguishes between different occurrences of a statement in the execution history; typically, it consists of a triple (input, occurrence of a statement, variable).

So, dynamic slicing makes explicit assumption about a fixed input of a program, static slicing does not.

Computing Static Slicing

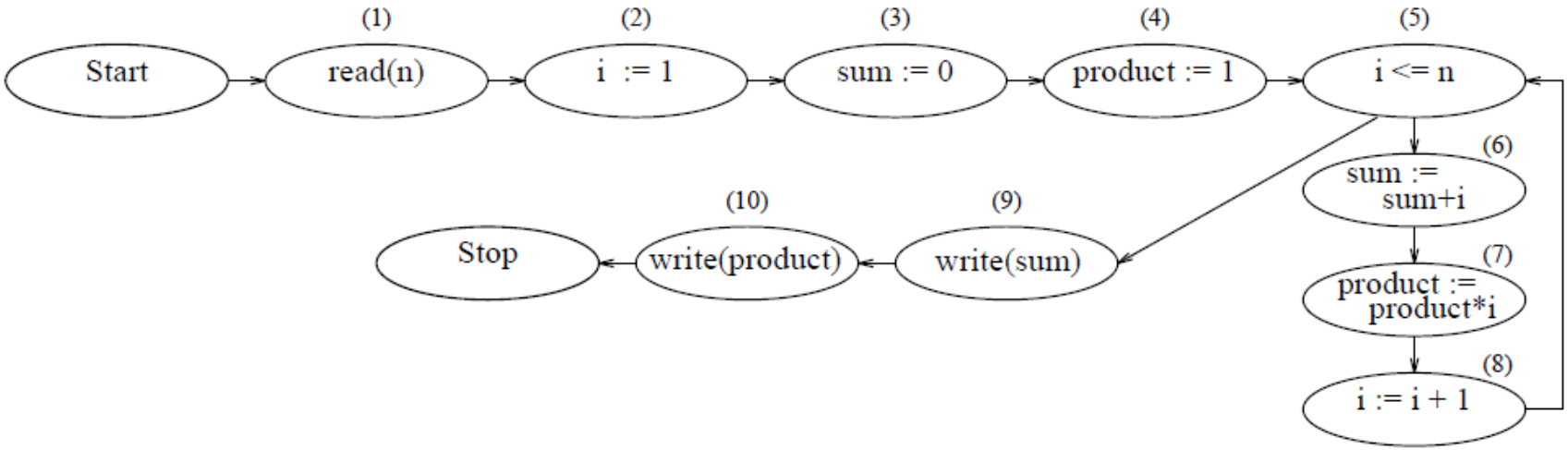


Figure 3: CFG of the example program of Figure 1 **(a)**.

Computing Static Slicing

Following Weiser's dataflow equations:

For each edge $i \rightarrow_{\text{CFG}} j$ in the CFG:

$$R_C^0(i) = R_C^0(i) \cup \{ v \mid v \in R_C^0(j),\ v \notin \text{DEF}(i) \} \cup \{ v \mid v \in \text{REF}(i),\ \text{DEF}(i) \cap R_C^0(j) \neq \emptyset \}$$

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

Figure 5: Equations for determining *directly* relevant variables and statements.

$$B_C^k = \{ b \mid \exists i \in S_C^k,\ i \in \text{INFL}(b) \}$$

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

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

Figure 6: Equations for determining *indirectly* relevant variables and statements.

Computing Static Slicing

Following Weiser's dataflow equations:

NODE #	DEF	REF	INFL	R_C^0	R_C^1
1	{ n }	∅	∅	∅	∅
2	{ i }	∅	∅	∅	{ n }
3	{ sum }	∅	∅	{ i }	{ i, n }
4	{ product }	∅	∅	{ i }	{ i, n }
5	∅	{ i, n }	{ 6, 7, 8 }	{ product, i }	{ product, i, n }
6	{ sum }	{ sum, i }	∅	{ product, i }	{ product, i, n }
7	{ product }	{ product, i }	∅	{ product, i }	{ product, i, n }
8	{ i }	{ i }	∅	{ product, i }	{ product, i, n }
9	∅	{ sum }	∅	{ product }	{ product }
10	∅	{ product }	∅	{ product }	{ product }

Table 1: Results of Weiser's algorithm for the example program of Figure 1 **(a)** and slicing criterion (10, { product }).

$$S_C^0 = \{2, 4, 7, 8\}$$
$$B_C^0 = \{5\}$$
$$S_C^1 = \{1, 2, 4, 5, 7, 8\}$$

Computing Dynamic Slicing

Following the dynamic flow concepts of Korel and Laski:

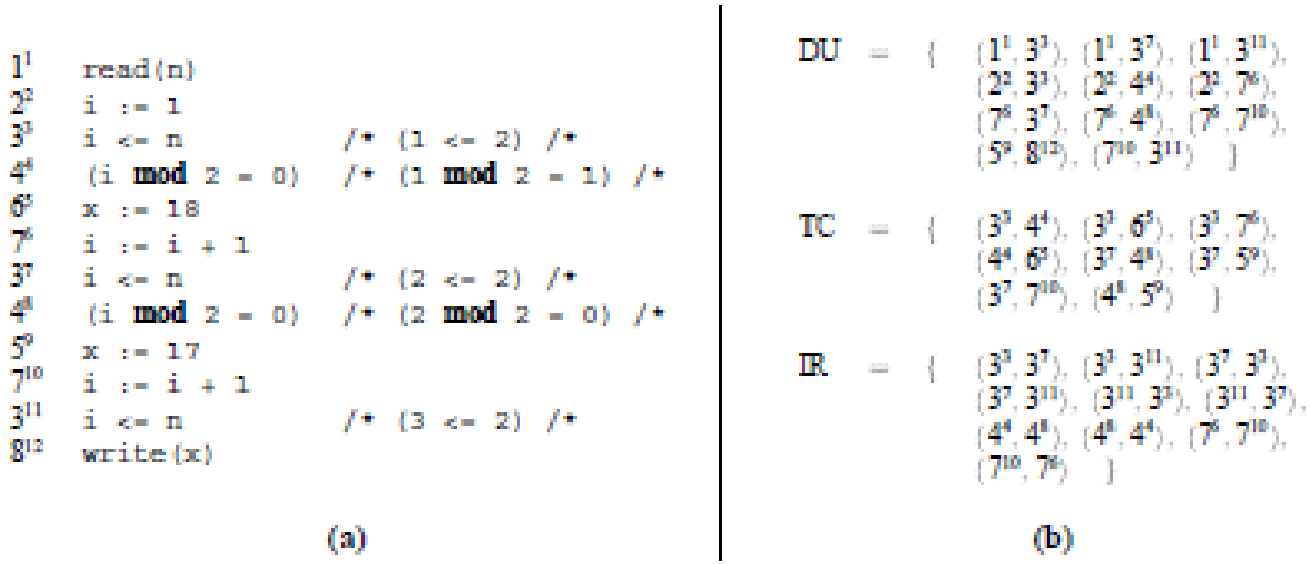


Figure 16: (a) Trajectory for the example program of Figure 2 (a) for input n = 2. (b) Dynamic Flow Concepts for this trajectory.

DU (Definition-Use) relation associates a use of a variable with its last definition.

TC (Test-Control) relation associates the most recent occurrence of a control predicate with the statement occurrences in the trajectory that are control dependent upon it.

IR (Symmetric Identity) relates occurrences of the same statement.

Computing Dynamic Slicing

Following the dynamic flow concepts of Korel and Laski:

$$
\begin{aligned}
S^{i+1} &= S^i \cup A^{i+1} \\
A^{i+1} &= \{X^p | X^p \ni S^i, (X^p, Y^q) \in (DU \cup TC \cup IR) \text{ for some } Y^q \in S^i\}
\end{aligned}
$$

The dynamic slicing w.r.t $(n = 2, 8^{12}, \{x\})$

$$
\begin{aligned}
A^0 &= \{5^9\} \\
A^1 &= \{3^7, 4^8\} \\
A^2 &= \{7^6, 1^1, 3^3, 3^{11}, 4^4\} \\
A^3 &= \{2^2, 7^{10}\} \\
S_C &= \{1^1, 2^2, 3^3, 4^4, 7^6, 3^7, 4^8, 5^9, 7^{10}, 3^{11}, 8^{11}\}
\end{aligned}
$$

Short Questions

1. How can you debug a program using slicing?
2. What slicing method should be used when you want to test why a variable is showing a value of '10' all the time?

Long Question

```
(1)    read(n);
(2)    i := 1;
(3)    while (i <= n) do
       begin
(4)      if (i mod 2 = 0) then
(5)        x := 17
         else
(6)        x := 18;
(7)      i := i + 1
       end;
(8)    write(x)
```

Static slicing criterion: $\{8, \{x\}\}$. Using Weiser's dataflow original definition of program slicing, compute all the directly relevant and indirectly relevant variables and statements for the above slicing criterion.

Reference to use to solve the above problem:
Frank Tip. A Survey of Program Slicing Techniques (See Section 3.1.1).