Program Analysis
Program Slicing (Part 2)
Outline

1. Introduction
2. Static Slicing
3. Thin Slicing
4. Dynamic Slicing

Mostly based on these papers:

- *Program Slicing*, Weiser., IEEE TSE, 1984
- *Thin Slicing*, Sridharan et al., PLDI 2007
- *Dynamic Program Slicing*, Agrawal and Horgan, PLDI 1990
Static Program Slicing

- Introduced by Weiser
  (IEEE TSE, 1984)

- Various algorithms to compute slices

- Here: Graph reachability problem based on program dependence graph
Program Dependence Graph

Directed graph representing the **data and control dependences** between statements

- **Nodes:**
  - Statements
  - Predicate expressions

- **Edges:**
  - Data flow dependences: One edge for each definition-use pair
  - Control flow dependences
Variable Definition and Use

- A **variable definition** for a variable $v$ is a basic block that assigns to $v$
  - $v$ can be a local or global variable, parameter, or property

- A **variable use** for a variable $v$ is a basic block that reads the value of $v$
  - In conditions, computations, output, etc.
A **definition-clear path** for a variable $v$ is a path $n_1, \ldots, n_k$ in the CFG such that

- $n_1$ is a variable definition for $v$
- $n_k$ is a variable use for $v$
- No $n_i$ (1 < $i$ ≤ $k$) is a variable definition for $v$
- $n_k$ may be a variable definition if each assignment to $v$ occurs after a use

Note: Def-clear paths do **not** go from entry to exit (in contrast to our earlier definition of paths)
Definition-Use Pair

A definition-use pair (DU-pair) for a variable \( v \) is a pair of nodes \( (d, u) \) such that there is a definition-clear path \( d, \ldots, u \) in the CFG.
Example: Data Flow Dependences

<table>
<thead>
<tr>
<th>Def</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```javascript
1 var n = readInput();
2 var i = 1;
3 var sum = 0;
4 var prod = 1;
5 while (i <= n) {
6   sum = sum + i;
7   prod = prod * i;
8   i = i + 1;
9 } console.log(sum);
10 console.log(prod);
```
Control Flow Dependences

- **Post-dominator:**
  Node $n_2$ (strictly) post-dominates node $n_1$ ($\neq n_2$) if every path $n_1, \ldots, \text{exit}$ in the control flow graph contains $n_2$.
Control Flow Dependences

- **Post-dominator:**
  Node $n_2$ (strictly) post-dominates node $n_1$ ($\neq n_2$) if every path $n_1, ..., \text{exit}$ in the control flow graph contains $n_2$.

- **Control dependence:**
  Node $n_2$ is control-dependent on node $n_1 \neq n_2$ if
  - there exists a control flow path $P = n_1, ..., n_2$ where $n_2$ post-dominates any node in $P$ (excluding $n_1$), and
  - $n_2$ does not post-dominate $n_1$.
Example: Control Flow Dependencies

1) Strict post-dominators

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2) Control dependences: 6 is control-dependent on 5

7
8
Computing Slices

Given:

- Program dependence graph $G_{PD}$
- Slicing criterion $(n, V)$, where $n$ is a statement and $V$ is the set of variables defined or used at $n$

Slice for $(n, V)$:

All statements from which $n$ is reachable (i.e., all statements on which $n$ depends)
Example: Program Dependence Graph

```javascript
1 var n = readInput();
2 var i = 1;
3 var sum = 0;
4 var prod = 1;
5 while (i <= n) {
6   sum = sum + i;
7   prod = prod * i;
8   i = i + 1;
9 }
10 console.log(sum);
11 console.log(prod);
```

- data dep.
- control flow dep.

slice (9, {sum})
= \{ n \mid reachable(n, 9) \}
= \{ 1, 2, 3, 5, 6, 8, 9 \}
Quiz

```javascript
var x = 1;   // 1
var y = 2;   // 2
if (x < y) { // 3
    y = x;   // 4
}
var z = x;   // 5
```

Draw the PDG and compute \( \text{slice}(5, \{z\}) \).

What is the sum of

- the number of nodes,
- the number of edges, and
- the number of statements in the slice?