Program Testing and Analysis:
Program Slicing (Part 1)

Dr. Michael Pradel
Software Lab, TU Darmstadt
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
Program Slicing

Extract an **executable subset of a program** that (potentially) **affects the values at a particular program location**

- **Slicing criterion** = program location + variable
- An observer focusing on the slicing criterion **cannot distinguish** a run of the program from a run of the slice
Example

```javascript
var n = readInput();
var i = 1;
var sum = 0;
var prod = 1;
while (i <= n) {
    sum = sum + i;
    prod = prod * i;
    i = i + 1;
}
console.log(sum);
console.log(prod);
```
Example

```javascript
var n = readInput();
var i = 1;
var sum = 0;
var prod = 1;
while (i <= n) {
    sum = sum + i;
    prod = prod * i;
    i = i + 1;
}
console.log(sum);
console.log(prod);
```

Slice for value of `sum` at this statement?
Example

```javascript
var n = readInput();
var i = 1;
var sum = 0;
var prod = 1;
while (i <= n) {
    sum = sum + i;
    prod = prod * i;
    i = i + 1;
}
console.log(sum);
console.log(prod);
```

Slice for value of sum at this statement?
Example

var n = readInput();
var i = 1;
var sum = 0;
var prod = 1;
while (i <= n) {
    sum = sum + i;
    prod = prod * i;
    i = i + 1;
}
console.log(sum);
console.log(prod);

Slice for value of prod at this statement
Example

```javascript
var n = readInput();
var i = 1;
var sum = 0;
var prod = 1;
while (i <= n) {
    sum = sum + i;
    prod = prod * i;
    i = i + 1;
}
console.log(sum);
console.log(prod);
```

Slice for value of `n` at this statement
Why Do We Need Slicing?

Various applications, e.g.

- **Debugging**: Focus on parts of program relevant for a bug
- **Program understanding**: Which statements influence this statement?
- **Change impact analysis**: Which parts of a program are affected by a change? What should be retested?
- **Parallelization**: Determine parts of program that can be computed independently of each other
Slicing: Overview

Forward vs. backward
■ Backward slice (our focus): Statements that influence the slicing criterion
■ Forward slice: Statements that are influenced by the slicing criterion

Static vs. dynamic
■ Statically computing a minimum slice is undecidable
■ Dynamically computed slice focuses on particular execution/input
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
Example: Data Flow Dependences

```javascript
var n = readInput();
var i = 1;
var sum = 0;
var prod = 1;
while (i <= n) {
    sum = sum + i;
    prod = prod * i;
    i = i + 1;
}
console.log(sum);
console.log(prod);
```
Control Flow Dependences

- Post-dominator:
  Node $n_2$ (strictly) post-dominates node $n_1 (\neq n_2)$
  if every path $n_1, \ldots, exit$ 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: Post-dominators

```javascript
var n = readInput();
var i = 1;
var sum = 0;
var prod = 1;
while (i <= n) {
    sum = sum + i;
    prod = prod * i;
    i = i + 1;
}
console.log(sum);
console.log(prod);
```

<table>
<thead>
<tr>
<th></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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></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>X</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>X</td>
<td>X</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>X</td>
<td>X</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>X</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>

**Control-dependence:**

6 is control-dep. on 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)
```javascript
var n = readInput();
var i = 1;
var sum = 0;
var prod = 1;
while (i <= n) {
    sum = sum + i;
    prod = prod * i;
    i = i + 1;
}
console.log(sum);
console.log(prod);
```

- **data dep.**
- **control dep.**

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

```
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?
1 \ var x = 1
2 \ var y = 2
3 \ if \ (x < y) \ {
4 \ \ \ \ y = x
5 \ }
6 \ var z = x

\[
\begin{array}{c}
\text{slice (5)} = \{1, 5\} \\
\text{Solution: 5 nodes + 5 edges + 2 statements = 12}
\end{array}
\]
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