Program Analysis

Program Slicing (Part 4)
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
Dynamic Slicing

- Various definitions
  Here: Agrawal & Horgan, PLDI 1990

- **Dynamic slice**: Statements of an execution that must be executed to give a variable a particular value
  - For an execution, i.e., a particular input
  - Slice for one input may be different from slice for another input

- Useful, e.g., for debugging: Get a reduced program that leads to the unexpected value
Dynamic Slice (Simple Approach)

- **Given:** *Execution history*
  - Sequence of PDG nodes that are executed

- **Slice for statement $n$ and variable $v$:**
  - Keep PDG nodes only if they are in history
  - Use static slicing approach (= graph reachability) on reduced PDG
Example 1

```javascript
var x = readInput();
if (x < 0) {
    y = x + 1;
    z = x + 2;
} else {
    if (x === 0) {
        y = x + 3;
        z = x + 4;
    } else {
        y = x + 5;
        z = x + 6;
    }
}
console.log(y);
console.log(z);
```
**Example: Dynamic Slicing (Simple Approach)**

```javascript
var x = readInput();
if (x < 0) {
    y = x + 1;
    z = x + 2;
} else {
    if (x === 0) {
        y = x + 3;
        z = x + 4;
    } else {
        y = x + 5;
        z = x + 6;
    }
}
console.log(y);
console.log(z);
```

Input: -1
History: 1, 2, 3, 4, 10, 11

PDG:

- Node 1
- Node 2
- Node 3
- Node 4
- Node 5
- Node 6
- Node 7
- Node 8
- Node 9
- Node 10
- Node 11

O... executed
O... slice (10, 3y3)
Example 2: Quiz

```javascript
var n = readInput(); // 1
var z = 0; // 2
var y = 0; // 3
var i = 1; // 4
while (i <= n) { // 5
    z = z + y; // 6
    y = y + 1; // 7
    i = i + 1; // 8
}
console.log(z); // 9
```
Example 2 (Quiz)

Input: 1
History: 1, 2, 3, 4, 5, 6, 7, 8, 5, 9

```javascript
var n = readInput();
var z = 0;
var y = 0;
var i = 1;

while (i <= n) {
    z = z + y;
    y = y + 1;
    i = i + 1;
}

console.log(z);
```

0.. in history
O.. shed (9, [2])

[Diagram of a directed graph with nodes and arrows indicating the flow of execution]
Limitations of Simple Approach

- **Multiple occurrences** of a single statement are represented as a **single PDG node**

- **Difference occurrences** of a statement may have different dependences
  - All occurrences get **conflated**

- **Slices** may be **larger than necessary**
Dynamic Slices (Revised Approach)

Dynamic dependence graph
- Nodes: Occurrences of nodes of static PDG
- Edges: Dynamic data and control flow dependences

Slice for statement $n$ and variables $V$ that are defined or used at $n$:
- Compute nodes $S_{dyn}$ that can reach any of the nodes that represent occurrences of $n$
- Slice = statements with at least one node in $S_{dyn}$
Example 2 (revised approach)

Input: 7
History: 1, 2, 3, 4, 5, 6, 7, 8, 5, 9

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

→ data

→ control

O.. slice (9, {2})
Discussion: Dynamic Slicing

- May yield a program that, if executed with another input, does not give the same value for the slicing criterion than the original program.

- Instead: Focuses on isolating statements that affect a particular value.
  - Useful, e.g., for debugging and program understanding.

- Other approaches exist, see F. Tip’s survey (1995) for an overview.