Program Analysis

Program Slicing (Part 3)

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Winter 2020/2021
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
Thin Slicing: Overview

■ Challenge: Static slices are often very large
  □ Worst case: Entire program
  □ Too large for common debugging and program understanding tasks

■ Main reason: Aims at an executable program
  □ But: Not needed for many tasks

■ Idea: Heuristically focus on statements needed for common debugging tasks
  → Thin slice

■ Let user expand the thin slice on demand
Thin Slicing: Definition

- **Statement directly uses** a memory location if it uses it for some computation other than pointer dereference
  - Example: \( x \cdot f + y \) uses \( x \) for pointer dereference and directly uses \( y \)

- **Dependence graph** \( G \) for thin slicing:
  Data dependences computed based on direct uses only

- **Thin slice**: Statements reachable from criterion’s statement via \( G \)
Example: Thin Slicing

```javascript
var x = {};
var z = x;
var y = {};
var w = x;
  w.f = y;
if (w === z) {
  var v = z.f;  // criterion
}
```

- Direct data dep.
- Data dep. only for pointer deref. (ignored)
- Control flow dep. (ignored)

---

**Dependence graph**

1 -> 4 -> 2 -> 6
3 -> 5 -> 7

- Traditional slice
  - All statements
- Thin slice
Expanding Thin Slices

- Thin slices include "producer statements" but exclude "explainer statements"
  - Why do heap accesses read/write the same object?
  - Why can this producer execute?
- Most explainers are not useful for common tasks
- Expose explainers on demand via incremental expansion
Example: Thin Slicing

```javascript
var x = {};  
var z = x;  
var y = {};  
var w = x;  
w.f = y;  
if (w === z) {  
    var v = z.f; // criterion
    }  
```  

- direct data dep.
- data dep. only for pointer deref. (ignored)
- control flow dep. (ignored)

Dependence graph

1 → 4 → 2 → 6

3 → 5 → 7

- Traditional slice
  - All statements

- Thin slice
  - On demand expansion, e.g.,
    "Why are w and z aliases?"
Evaluation and Results

- **Simulate** developer effort for **bug finding**
  - Set of known bugs that crash the program (and their root causes)
  - Assume that developer inspects statements with breadth-first search on PDG, starting from crash point
  - Count inspected statements with traditional and thin slice

- **Results:**
  - Mean of **12 inspected statements** per thin slice
  - Overall, **3.3x fewer inspected statements**