Program Analysis

Program Slicing

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2023/2024

Warm-up Quiz

What does the following JavaScript code print?

```
let x = [1, 2, 1];
let y = [2, 3, 2];
let z;
if (Number.MIN_VALUE > 0) {
    z = x + y;
} else {
    z = x[y.length - 1];
}
console.log(z);
```

Warm-up Quiz

What does the following JavaScript code print?

```
let x = [1, 2, 1];
let y = [2, 3, 2];
let z;
if (Number.MIN_VALUE > 0) {
    z = x + y;
} else {
    z = x[y.length - 1];
}
console.log(z);
```

Result: 1,2,12,3,2

Warm-up Quiz

What does the following JavaScript code print?

```
let x = [1, 2, 1];
let y = [2, 3, 2];
let z;
if (Number.MIN_VALUE > 0) {
    z = x + y;
} else {
    z = x[y.length - 1];
}
console.log(z);
```

Result: 1,2,12,3,2

`MIN_VALUE`: Smallest *positive* value that can be representated within floating point precision

(Use `NEGATIVE_INFINITY` for the overall smallest value)

Warm-up Quiz

What does the following JavaScript code print?

```
let x = [1, 2, 1];
let y = [2, 3, 2];
let z;
if (Number.MIN_VALUE > 0) {
    z = x + y;
} else {
    z = x[y.length - 1];
}
console.log(z);
```

Adding two arrays: Convert both to string and concatenate

Result: 1,2,12,3,2

Outline

1. **Introduction**
2. **Static Slicing**
3. **Dynamic Slicing**

Mostly based on these papers:

- *Program Slicing*, Weiser., IEEE TSE, 1984
- *Dynamic Program Slicing*, Agrawal and Horgan, PLDI 1990
- *A Survey of Program Slicing Techniques*, Tip, J Prog Lang 1995

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

```
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

```
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? (pointing to `console.log(sum);`)

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 `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

```
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

Outline

1. **Introduction**
2. **Static Slicing** ←
3. **Dynamic Slicing**

Mostly based on these papers:

- *Program Slicing*, Weiser., IEEE TSE, 1984
- *Dynamic Program Slicing*, Agrawal and Horgan, PLDI 1990
- *A Survey of Program Slicing Techniques*, Tip, J Prog Lang 1995

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.

Definition-Clear Paths

A definition-clear path for a variable v is a path $n_1, .., n_k$ in the CFG such that

- n_1 is a variable definition for v
- n_k is a variable use for v
- No n_i $(1 < i \leq k)$ 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

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 $d, .., u$ in the CFG

Example: Data-Flow Dependencies

```
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);
```

Def \ Use	1	2	3	4	5	6	7	8	9	10
1					X					
2					X	X	X	X		
3						X			X	
4							X			X
5										
6						X			X	
7							X			X
8					X	X	X	X		
9										
10										

Control Flow Dependences

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

1) Strict post-dominators

n_1 \ n_2	1	2	3	4	5	6	7	8	9	10
1		X	X	X	X				X	X
2			X	X	X				X	X
3				X	X				X	X
4					X				X	X
5									X	X
6					X		X	X	X	X
7					X			X	X	X
8					X				X	X
9										X
10										

2) Control-flow deps: 6 is control-dependent on 5
7 — " — 5
8 — " — 5

Computing Slices

Given:

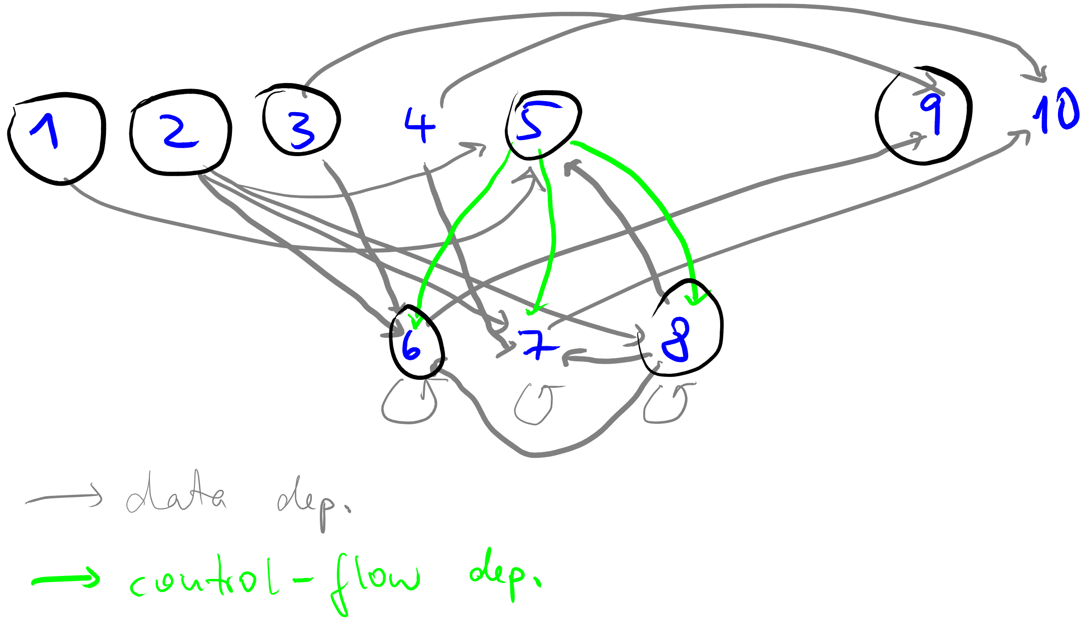
- Program dependence graph G_{PD}
- Slicing criterion (n, V), where n is a statement and V is the set of variables defined or used at n

Slice for (n, V):

All statements from which n is reachable
(i.e., all statements on which n depends)

Example: Program Dependency Graph

```
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);
```



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

Quiz

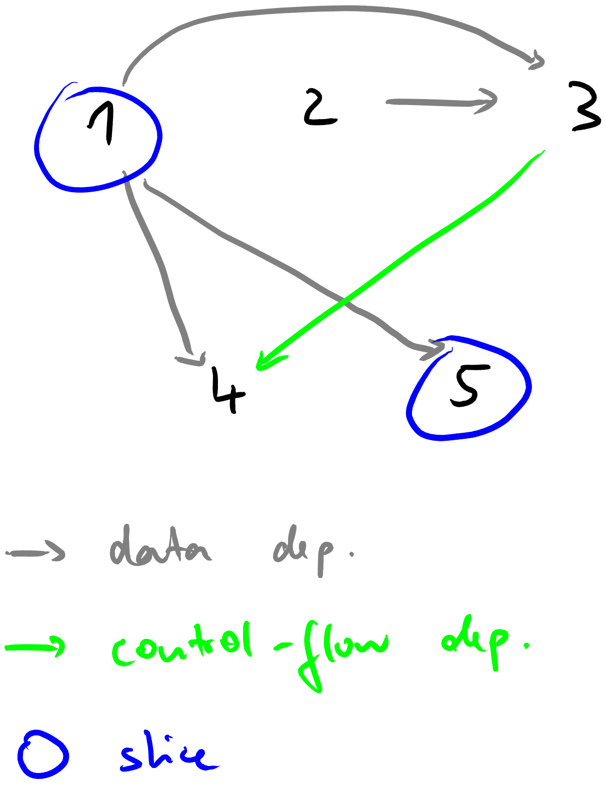
```
var x = 1;      // 1
var y = 2;      // 2
if (x < y) {    // 3
  y = x;        // 4
}
var z = x;      // 5
```

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

What is the sum of

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

Quiz:



Outline

1. **Introduction**
2. **Static Slicing**
3. **Dynamic Slicing** ←

Mostly based on these papers:

- *Program Slicing*, Weiser., IEEE TSE, 1984
- *Dynamic Program Slicing*, Agrawal and Horgan, PLDI 1990
- *A Survey of Program Slicing Techniques*, Tip, J Prog Lang 1995

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 only those PDG nodes that are in history
  - Use static slicing approach (= graph reachability) on reduced PDG

Example 1

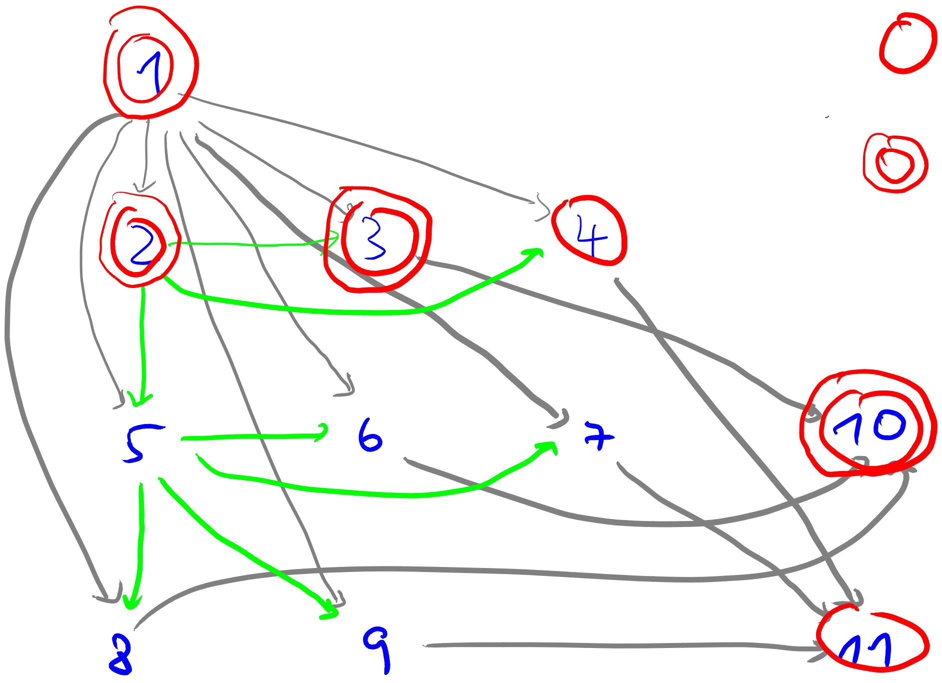
```
var x = readInput();
if (x < 0) {
  y = x + 1;
  z = x + 2;
} else {
  if (x === 0) {
    y = x + 3;
    z = x + 4;
  } else {
    y = x + 5;
    z = x + 6;
  }
}
console.log(y);
console.log(z);
```

Example: Dynamic Slicing (Simple Approach)

Input: -1

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

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



O .. executed

◎ .. slice(10, {y})

→ data

→ control

Example 2: Quiz

```
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

```
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
```

Draw the PDG and compute the dynamic slice for statement 9 and variable `z`, with input `n=1`.

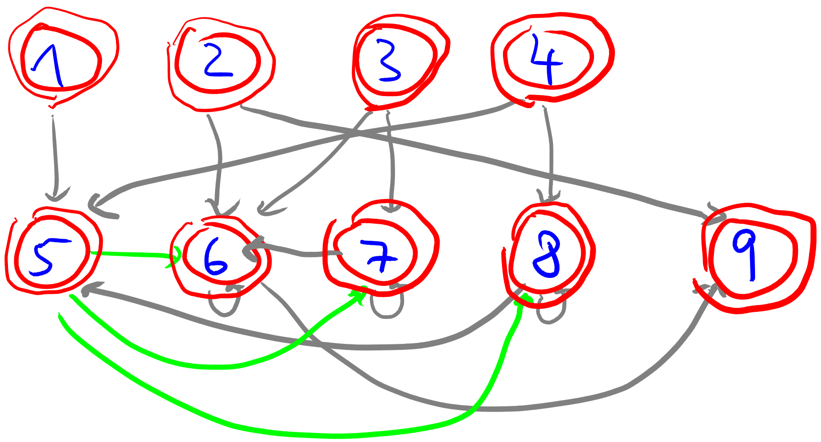
How many statements are in the slice?

Example 2 (Quiz)

Input: 1

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

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



○ .. history

◎ .. slice(9, {z})

→ data

→ control

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 Slice (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:

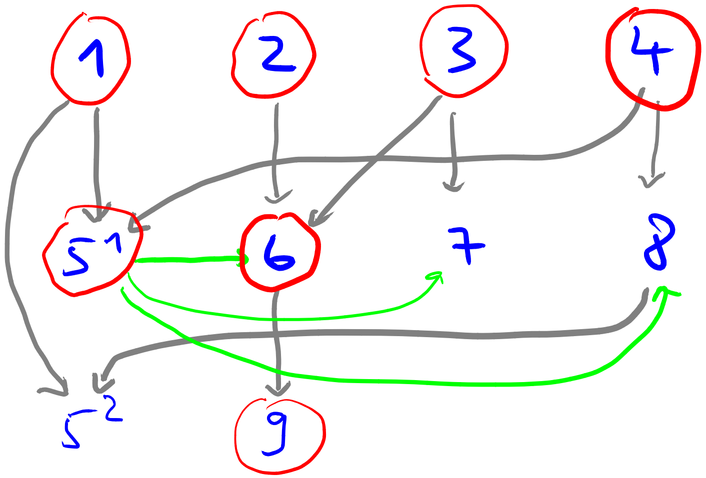
- Compute nodes S_{dyn} that can reach any of the nodes that represent occurrences of n
- Slice = statements with at least one node in S_{dyn}

Example 2 (Revised Approach)

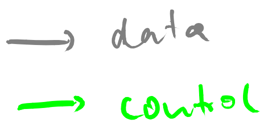
Input: 1

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

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



○ .. slice(9, {z})



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

Outline

1. **Introduction**
2. **Static Slicing**
3. **Dynamic Slicing** ✓

Mostly based on these papers:

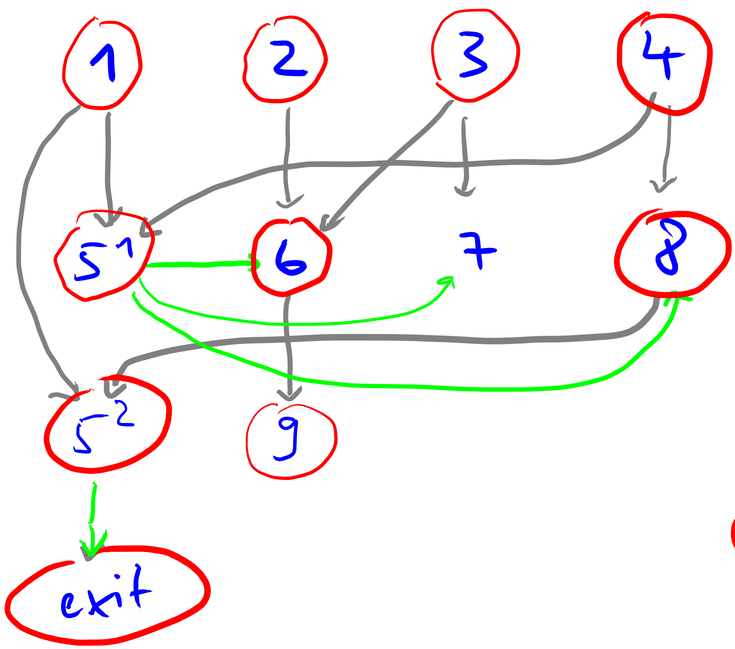
- *Program Slicing*, Weiser., IEEE TSE, 1984
- *Dynamic Program Slicing*, Agrawal and Horgan, PLDI 1990
- *A Survey of Program Slicing Techniques*, Tip, J Prog Lang 1995

Example 2 (Revised Approach)

Input : 1

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

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



⟶ data

⟶ control

◯ .. slice(9, {z})