Program Analysis

Introduction of Course Project

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2021/2022

Warm-up Quiz

What does the following code print?

```
function d(x) {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve(x * 2);
        }, 2000);
    });
  }

d(5).then((r) => {
    console.log(r);
});
```

5 **10** **undefined** **Something else**

Warm-up Quiz

What does the following code print?

```
function d(x) {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve(x * 2);
        }, 2000);
    });
}

d(5).then((r) => {
    console.log(r);
});
```

(but only after waiting for two seconds)

5 **10** undefined Something else

Warm-up Quiz

What does the following code print?

```
function d(x) {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve(x * 2);
        }, 2000);
    });
}

d(5).then((r) => {
    console.log(r);
});
```

Promise: Represents a result that is not yet complete

5 **10** undefined Something else

Warm-up Quiz

What does the following code print?

```
function d(x) {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve(x * 2);
        }, 2000);
    });
}

d(5).then((r) => {
    console.log(r);
});
```

Wait 2,000 milliseconds and then return the promise's result

5 | **10** | undefined | Something else

Warm-up Quiz

What does the following code print?

```
function d(x) {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve(x * 2);
        }, 2000);
    });
}

d(5).then((r) => {
    console.log(r);
});
```

Wait for the promise to resolve and then use its result

5 **10** undefined Something else

Goal

Design and implement dynamic slicing

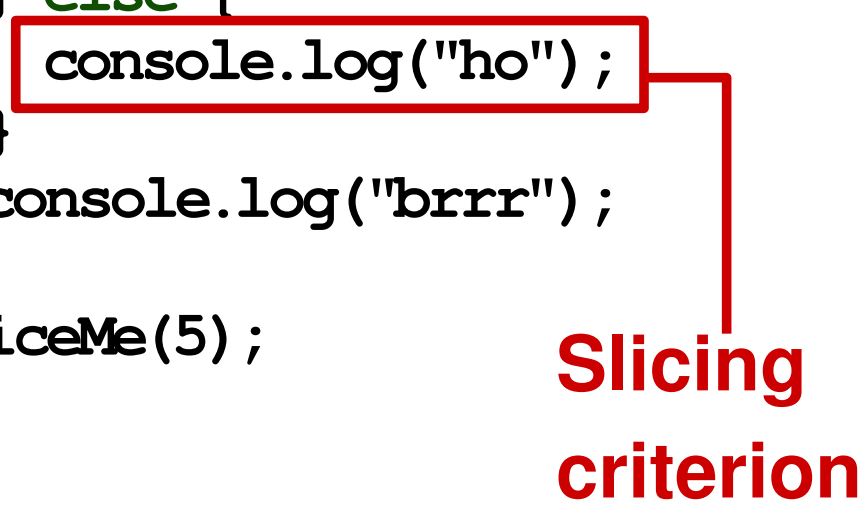
- Input:
  - Executable program with all inputs
  - Slicing criterion
- Output:
  - Reduced program that yields same behavior w.r.t. slicing criterion (for same input)

Example

```
function sliceMe(n) {
  var x = n + 1;
  if (x == 5) {
    console.log("hey");
  } else {
    console.log("ho");
  }
  console.log("brrr");
}
sliceMe(5);
```

Example

```
function sliceMe(n) {
  var x = n + 1;
  if (x == 5) {
    console.log("hey");
  } else {
    console.log("ho");
  }
  console.log("brrr");
}
sliceMe(5);
```



Example

```
function sliceMe(n) {
  var x = n + 1;
  if (x == 5) {
    console.log("hey");
  } else {
    console.log("ho");
  }
  console.log("brrr");
}
sliceMe(5);
```

Slicing criterion

```
function sliceMe(n) {
  var x = n + 1;
  if (x == 5) {
  } else {
    console.log("ho");
  }
}
sliceMe(5);
```

Example

```
function sliceMe(n) {
  var x = n + 1;
  if (x == 5) {
    console.log("hey");
  } else {
    console.log("ho");
  }
  console.log("brrr");
}
sliceMe(5);
```

Slicing criterion

Example

```
function sliceMe(n) {
  var x = n + 1;
  if (x == 5) {
    console.log("hey");
  } else {
    console.log("ho");
  }
  console.log("brrr");
}
sliceMe(5);
```

Slicing criterion

```
function sliceMe(n) {
  console.log("brrr");
}
sliceMe(5);
```

Slicing Algorithms

Different algorithms differ in

- Precision: How small does the slice get?
- Efficiency: How long does the slicing take?
- Conceptual complexity

Objective: Smallest possible slice (i.e., as precise as possible), but still sound

- Soundness: All statements included to preserve behavior w.r.t. slicing criterion

Assumptions

Kind of programs to consider

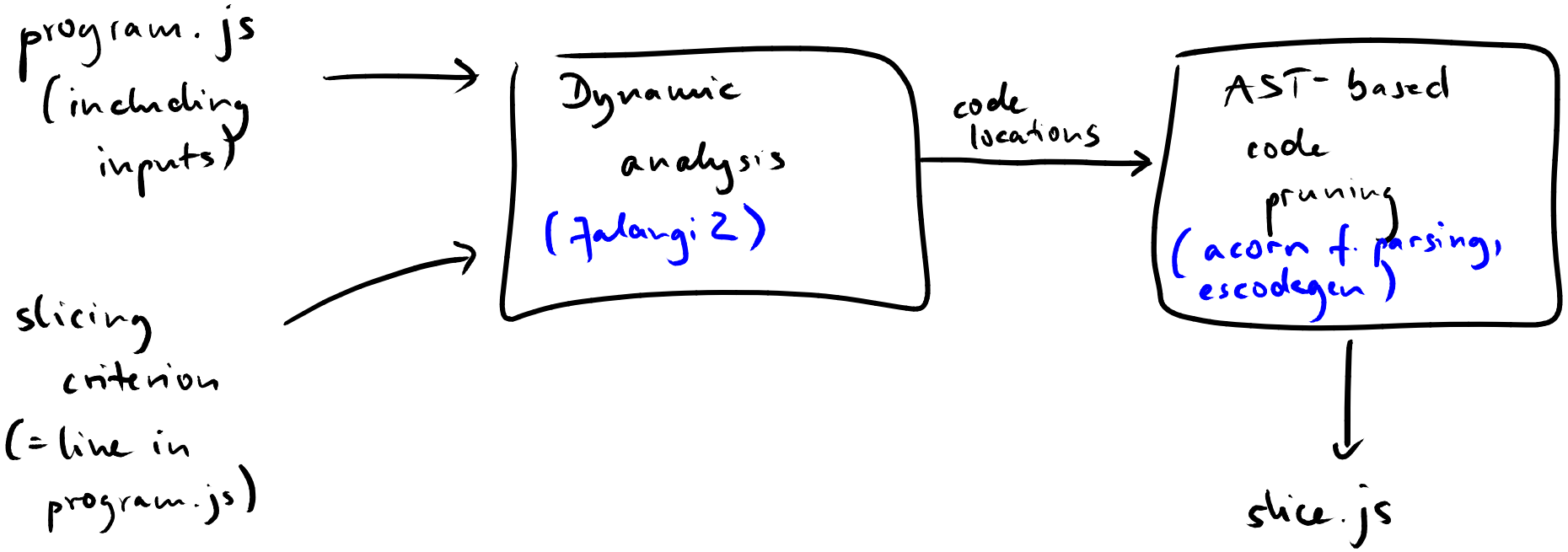
- Single function
- Single file: Defines the function and then calls it
- Slice should always keep all arguments to the sliced function (even if unused)
- Calls to other functions:
  - May return a value, which is data-dependent on arguments of the call
  - Otherwise, free of side-effects

Assumptions (2)

Subset of JavaScript to consider

- Language features until ECMAScript 5 (ES5)
- No calls to `eval`
- No `with` statements
- Each variable declared on a single line, i.e., no

```
var a, b, c;
```


program.js
(including inputs)
slicing criterion
(= line in program.js)
Dynamic analysis
(Jalangi 2)
code locations
AST-based code pruning
(acorn f. parsing, escodegen)
slice.js

Dynamic Analysis

- **Based on Jalangi framework**
- **Hooks/callbacks for different kinds of runtime events, e.g.,**
  - variable reads/writes
  - binary expressions
  - conditionals
- **Based on source-to-source instrumentation**

Tips on Jalangi

- **Rich framework that provides more than what you need**
- **Use "Analysis in node.js with on-the-fly instrumentation"**
  - Input: Program to analyze and the analysis itself
  - Instruments and then runs the program

Implementing Slicing

- **Track data-flow and control-flow dependencies at runtime**
  - Data flow: Whenever a new value gets computed, track dependency from inputs
  - Control flow: Whenever a control flow decision is made, track what it depends on

Location Information

- Every runtime event happens at some code location
- IID = unique identifier of location in original program (i.e., before instrumentation)
- Use it to determine which code is needed in the slice

AST-based Pruning of Code

- **Once locations to keep are known:**
  - Prune away remaining code
- **Implement it via AST transformation**
  - Parse
  - Manipulate
  - Pretty-print

Project Milestones

- **Milestone 1**
  - Simple Jalangi analysis
  - AST manipulation
- **Milestone 2**
  - Data-flow only slicing
- **Milestone 3**
  - Control-flow and data-flow

Milestone 1: Simple Jalangi Analysis

- **Goal: Prints values of variable writes**
  - Meta-level goal: Get familiar with Jalangi
- **Example:**

```
var x;
var y = 0;
x = 23;
if (x > 5) {
  y = x - 3;
}
console.log(y)
```

Milestone 1: Simple Jalangi Analysis

- **Goal: Prints values of variable writes**
  - Meta-level goal: Get familiar with Jalangi
- **Example:**

```
var x;
var y = 0;
x = 23;
if (x > 5) {
  y = x - 3;
}
console.log(y)
```

0

23

20

Milestone 1: AST Manipulation

- **Input: Code, line numbers**
- **Output: Subset of code**
- **Example:**

```
// lines to keep: 2, 3, 5
var x;
var y = 0;
x = 23;
if (x > 5) {
  y = x - 3;
}
console.log(y)
```

Milestone 1: AST Manipulation

- **Input: Code, line numbers**
- **Output: Subset of code**
- **Example:**

```
// lines to keep: 2, 3, 5
var x;
var y = 0;
x = 23;
if (x > 5) {
  y = x - 3;
}
console.log(y)
```

```
var y = 0;
x = 23;
y = x - 3;
```

Milestone 2

- Slicing based on data flow only
- Assume: Straightline code without control flow
- Example:

```
var x;
var y = 0;
x = 23;
var z = 5;
y = x - 3;
z = x++;
z = y * 3;
```

Milestone 2

- Slicing based on data flow only
- Assume: Straightline code without control flow
- Example:

```
var x;
var y = 0;
x = 23;
var z = 5;
y = x - 3;
z = x++;
z = y * 3;
```

Slicing criterion

Milestone 2

- Slicing based on data flow only
- Assume: Straightline code without control flow
- Example:

```
var x;
var y = 0;
x = 23;
var z = 5;
y = x - 3;
z = x++;
z = y * 3;
```

Slicing criterion

```
var x;
var y = 0;
x = 23;
y = x - 3;
```

Milestone 3

- Slicing based on both data flow and control flow
- Now, code may have branches, loops, etc.
- Example:

```
var x = 3;
if (x > -2) {
  console.log(x);
}
console.log(x);
```

Milestone 3

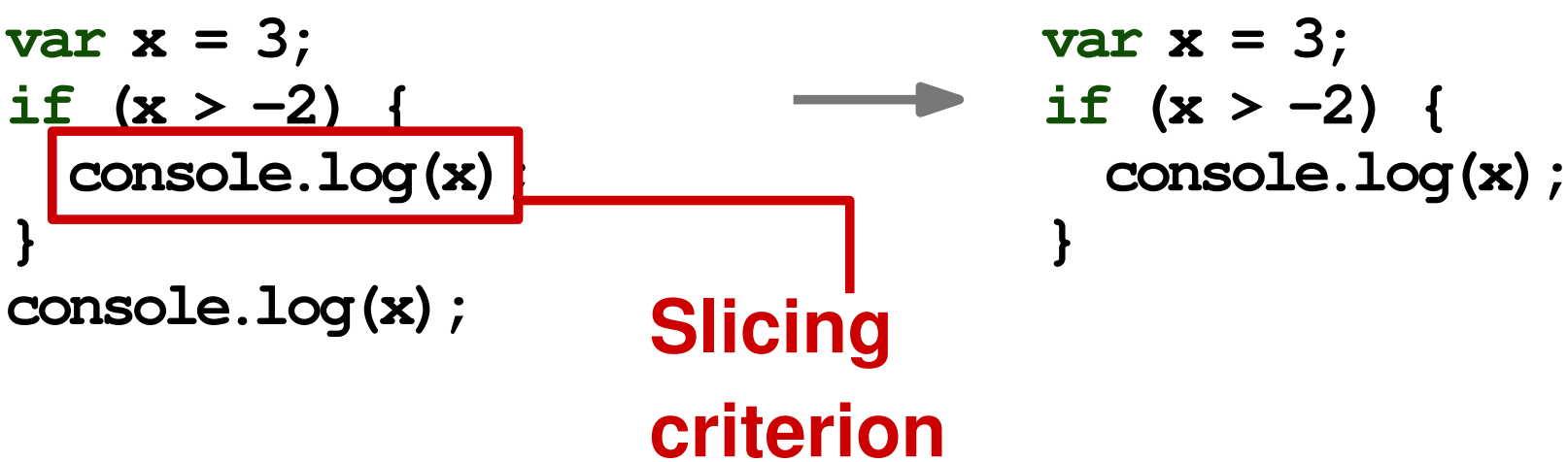
- Slicing based on **both data flow and control flow**
- Now, code may have branches, loops, etc.
- Example:

```
var x = 3;
if (x > -2) {
  console.log(x);
}
console.log(x);
```

Slicing criterion

Milestone 3

- Slicing based on both data flow and control flow
- Now, code may have branches, loops, etc.
- Example:



Scripts and Tests

Provided by us:

- **To-be-implemented script slice.js**
- **Test suite of programs to slice**
- **Script to run test suite**

Mentoring

- Each student gets a mentor
- Meet at least three times (once per milestone)
- Mentor assignment and meeting dates: Message in Ilias

Timeline

- **Milestone 1: Due in week of Dec 13–17**
- **Milestone 2: Due in week of Jan 10-15**
- **Milestone 3: Due in week of Jan 24-28**
- **Full project due: Feb 11**
  - Project report (up to 4 pages)
  - Your implementation
- **Oral presentation: Week of Feb 14–18**

Timeline

- **Milestone 1: Due in week of Dec 13–17**
- **Milestone 2: Due in week of Jan 10-15**
- **Milestone 3: Due in week of Jan 24-28**
- **Full project due: Feb 11**
  - Project report (up to 4 pages)
  - Your implementation
- **Oral presentation: Week of Feb 14–18**

Soft deadlines

Timeline

- **Milestone 1: Due in week of Dec 13–17**
- **Milestone 2: Due in week of Jan 10-15**
- **Milestone 3: Due in week of Jan 24-28**
- **Full project due: Feb 11**
  - Project report (up to 4 pages)
  - Your implementation
- **Oral presentation: Week of Feb 14–18**

Hard deadlines