Program Analysis

– Project Description, Winter Semester 2021/22 –

Prof. Dr. Michael Pradel, Aryaz Eghbali, Moiz Rauf

November 29, 2021

1 Introduction

Program slicing is a type of dynamic analysis that has multiple applications, one of which is locating the root cause of a bug. In slicing, a subprogram is extracted based on some slicing criterion to reduce the size of the code, which facilitates the downstream tasks. There are multiple slicing algorithms, and you have seen some of them during the lecture.

2 Project Description

The goal of this project is to implement **dynamic backward slicing** for JavaScript. The exact choice of the slicing algorithm is up to you, and in particular, you are not restricted to algorithms discussed in the lecture. The project will be based on Jalangi[1], a general purpose framework for dynamic analysis of JavaScript. Your implementation should also be in JavaScript. Given a JavaScript program and a slicing criterion, the final output of your analysis should keep only the code needed for the slice, e.g., by removing subtrees from the AST that are not part of the slice. To reduce the given source code into the slice, tools like acorn[2] and escodegen[3] will be helpful for parsing and generating code, respectively.

2.1 Scope Limitations

To help students focus on the main analysis and avoid the hassle of dealing with all of the language features of JavaScript, we have decided to narrow down the scope of programs that your implementation should be able to handle:

- You will implement an intra-procedural analysis, which means your analysis is limited to the execution inside a function. You can assume that any function that is called during the execution does not have any side-effects, i.e., it does not change the values of any variable or object and only might return a value.
- You are not required to support JavaScript features after ECMAScript 5th edition (ES5).
- Calls to `eval` and `with` statements are considered out of the scope of this project, and will not appear in our test cases.
- Each variable is declared on a separate line, i.e., multiple variable declarations like `var a, b, c;` will not appear in the test cases.
- When slicing the function, all function arguments remain intact, even if an argument is not needed in the slice.

[1] `https://github.com/Samsung/jalangi2`
[2] `https://github.com/acornjs/acorn`
[3] `https://github.com/estools/escodegen`

3 Milestones

The project is broken down into three milestones for better guidance throughout the semester. Each milestone has a progress meeting, in which each student meets with her/his mentor, presents the deliverable, and seeks help if needed. Please create a **private** GitHub repository for your project and give read access to your mentor. Some code templates have been prepared for each milestone, on top of which you should implement your solution. Please adhere to the output format shown in the provided test cases, as the output of your program will be automatically tested.

3.1 Milestone 1 (Progress Meeting in Week of Dec 13–17, 2021)

In the first part of the project you will familiarize yourself with Jalangi by implementing a simple dynamic analysis. Furthermore, you will also implement multiple methods that are required for manipulating the source code and for printing the final result. On the progress meeting of this milestone, course participants are expected to have implemented the following:

1. A dynamic analysis in Jalangi that each time a variable is being written to, prints the value being written. For example, in the following program

   ```
   function sliceMe() {
       var x = 5;
       var y;
       console.log("Hello World");
       if (x < 10)
           x = x + 5;
       y = 0;
       return y;
   }
   sliceMe();
   ```

   the analysis should output

   ```
   5
   10
   0
   ```

   As a starting point, we suggest adapting this tutorial.

2. A function that given the path of a JavaScript file and a set of code locations to keep, removes all other parts of the code from the file. To this end, the function should parse the code, e.g., with acorn, manipulate the AST, and then pretty print the code, e.g., with escodegen. For example, given the above source code and lines `[1, 2, 5, 6, 9, 10]`, the function should produce

   ```
   function sliceMe() {
       var x = 5;
       if (x < 10)
           x = x + 5;
   }
   sliceMe();
   ```

3.2 Milestone 2 (Progress Meeting in Week of Jan 10–15, 2022)

The expectation for this milestone is that each student has implemented dynamic program slicing based on data-flow dependencies. For the following example,

```
function sliceMe() {
    var x;
    var y;
    x = 1;
```

```
    y = 2;
    x = x+y;
    y += 2;
    return y; // slicing criterion
}
sliceMe();
```

the correct output is

```
function sliceMe() {
    var y;
    y = 2;
    y += 2;
    return y;
}
sliceMe();
```

Each input for this milestone consists of an input JavaScript file, which is the program to be sliced, and a line number, which gives the slicing criterion. Note that all variables used in the given line are part of the slicing criterion. You can safely assume that in this milestone the input code consists only of functions with a single control flow path, i.e., no branches.

3.3 Milestone 3 (Progress Meeting in Week of Jan 24–28, 2022)

It is expected that each course participant has implemented the complete slicing algorithm, including both data-flow and control-flow dependencies. For example, the output of your implementation for

```
function sliceMe() {
    var x;
    var y;
    var z;
    x = 1;
    y = 2;
    z = 3;
    if (x < 4)
        y += 2;
    if (x > 0)
        z -= 5;
    return y; // slicing criterion
}
sliceMe()
```

should be

```
function sliceMe() {
    var x;
    var y;
    x = 1;
    y = 2;
    if (x < 4)
        y += 2;
    return y;
}
sliceMe();
```

Similar to the second milestone, the input of your analysis is a JavaScript file and a line number, which indicate the program to be sliced and the slicing criterion, respectively.

4 Running and Testing Your Code

4.1 Dataset

Test Cases: We have provided some sample test cases for Milestones 2 and 3. The test cases are listed in a JSON file where each test case is an object with four fields: *inFile*, *outFile*, *lineNb*, *goldFile*, which indicate the input file path, the file path to write your slice into, the line number of the slicing criterion, and the file path with the expected slice, respectively. For example:

```
[
{"inFile": "a1_in.js", "outFile": "a1_predicted_out.js",
        "lineNb":5, "goldFile": "a1_out.js"},
{"inFile": "a2_in.js", "outFile": "a1_predicted_out.js",
        "lineNb":19, "goldFile": "a2_out.js"}
]
```

In addition to the provided test cases, we strongly recommend to test your code with additional test cases. To add test cases, you should create an input file, an expected output file, and add the test case details to the tests JSON file. The format of the input file should remain the same.
Your implementation will be evaluated with test cases that are not provided to you.

Javascript Code For consistency and to ease the task, all input code files follow this convention:

- There is a single function definition, with the function called `sliceMe`, which is the function to be sliced.
- The last statement in the file is a call to the `sliceMe` function.

The examples given above follow this convention.

4.2 Implementation & Execution

As part of the project repository you are provided with two scripts `slice.js` and `testRunner.js`. They provide a unified interface for execution and evaluation.

slice.js is entry point into your implementation of the slicing algorithm. This script runs your code on the input file and writes the sliced code into the output file. Please add your code into the $slice(inFile, outFile, lineNb)$ function. Apart from the signature of this method and the command-line arguments of the script, you can change the code according to your requirements.
The script takes an input file (`--inFile <input_file_path>`), a slice criteria line number (`--lineNb <line_number>`), and an output file (`--outFile <output_file_path>`). You can run the script using following command:

```
node slice.js --inFile <input_file_path> --outFile <output_file_path>\
--lineNb <line_number>
```

testRunner.js provides means to execute `slice.js` on a set of test cases. The script reads a `testCases.json` file, executes the `slice.js` script to produce output files, and finally evaluates the produced slices against a gold standard. The script takes an input file `--source <json_file_path>` as an argument. You can run the test runner using the following command:

```
node testRunner.js --source <json_input_file_path>
```

The test runner contains a function `compare(originalFile, predictedFile)` that compares the produced slice with the expected slice. If both do not match exactly, it compares the two code files using Levenshtein distance.

5 Installation Guidelines

Detailed instructions on how to install Jalangi2 is described in the installation guide. To ensure consistency please follow the version requirements for Jalangi, i.e.

1. Node.js LTS version v12
2. Python version between 2.7 and 3.0.

In addition to requirements for Jalangi, you would need the following npm packages for executing provided scripts: argparse, fast-levenshtein, acorn, escodegen.

6 Mentoring and How to Submit

Each student has a mentor, either Aryaz Eghbali (aryaz.eghbali@iste.uni-stuttgart.de) or Moiz Rauf (moiz.rauf@iste.uni-stuttgart.de). Students must meet their mentor at least three times during the semester, on the dates indicated for the milestones. In addition, students may consult their mentor to resolve any questions, to ensure progress in the right direction, and to help you submit a successful project in time. We also recommend to ask general questions in the Ilias forum.

The deadline for submitting the project is at 11:59 PM on February 11, 2022. This deadline is firm. The submission will be via Ilias and should include:

- A scientific report of at most four pages that summarizes the approach taken, the results obtained, and any shortcomings you are aware of.
- A .zip file with your implementation. The implementation must be usable via the scripts we provide as a template. Please do not change the interface of these scripts.

Each person must present the project on the week of February 14–18, 2022, in a short talk, followed by a question and answer session.

The project is individual, i.e., students must work by themselves and not share their solution with anyone else. Any form of collaboration that results in similar or identical solutions being submitted will be punished according to the usual rules for plagiarism.

Grading will be based on the following criteria:

Criterion	Contribution
Progress meetings and final presentation (progress made, clarity, illustration, quality of answers)	25%
Implementation (completeness, documentation)	25%
Results (soundness, reproducibility)	25%
Report (clarity, illustration, discussion and interpretation of the results)	25%