TECHNISCHE UNIVERSITÄT DARMSTADT

Course-Project: Finding Dynamic Dead Writes

Program Testing and Analysis WS 2017/2018

Kai-Nico Nase (2644914) and Nicolas Voigt (1289808)

supervised by
Dr. Michael Pradel
and
Jibesh Patra

February 19, 2018

Contents

List of Figures

1 Abstract

This report reflects the progress and results of a Course-Project called "Finding Dynamic Dead Writes" that is part of the subject "Program-Testing and Analysis" at TU-Darmstadt. The first part lists tasks of the task-sheet up that are to be performed. The section "Introduction" describes in general which first steps have been taken to solve the tasks together in teamwork and which tools have been used, to simplify many things. The following sections describe then the definition of a "Dead-Write", which initial tests were carried out and of which components our project consists of. Then the approach and the implementation or components of our code analysis, which is able to detect certain types of "Dead-Writes" and afterwards cleanly remove them will be presented. Then some sample programs are listed up that contain different types of dead-writes, which our written Jalangi-Code-Analysis should be able to cleanly remove. At the end, special problems that occurred during our work on the project will be explained, as well as the final results and an outlook.

2 Goals from the Tasks-Sheet

1. The goal of this project is to write a Jalangi [1] analysis that will find dead writes during the execution of JavaScript programs. The analysis should guarantee that some writes are dead for some possible executions of the program. Given a JavaScript program, the output of the project will be the program without the dead writes
2. Write few simple JavaScript programs that contain dead writes
3. Get familiar with Jalangi2 and design and implement a Jalangi based analysis that finds dynamic dead writes
4. Validate your implementation with the self-written programs
5. Parse each program using esprima to generate an AST. Traverse the AST using estraverse and delete the identify dead writes
6. For the final evaluation, use the JavaScript libraries Backbone, Backbone LayoutManager,BigNumber, Bootstrap Datepicker, Cal-Heatmap, Countdown, Underscore. These libraries have available tests. Run your analysis while executing these tests
7. Remove the dead writes and run the tests again to see if any of them fail.
8. Report your results using the following questions:
   - Overall, what percentage of writes are dead?
   - What is the difference in size (bytes) of the program after removing the dead writes.
   - Is there any noticeable performance improvements after removing the dead writes?

   Bonus points: Apart from the given packages, identify 5 node packages that have available tests and run your analysis on them.

3 Introduction

At the begin and after the first briefly meeting with our course-mentor, we discussed "live" first, how to configure our Test-Environment and how would be the best way to share the work, respectively "Who takes which part". Via GitHub, we set up a repository in order to keep our project-files always up to date. Due to the fact that we both live farther away from each other, we decided to hold our regular team meetings online via a free version of the Teamviewer[3] software. We also agreed at the beginning that we would like to create(for this report) diagrams or graphics with the web-based software DRAW.IO[2] so that it wouldn't be necessary to install any additional desktop-software such as e.g. MS Visio. After that, each of us installed a Linux-Image(Ubuntu) on a virtual-machine(VMWare[4]) and subsequently the respective NodeJS[5], which is necessary to use jalangi[1] without using a web-browser. Now that we both had familiarized ourselves with Jalangi, we thought about which Callback-Functions we could use from the "Standard-Jalangi-Code-Template" to write the code analysis that Dead-Writes should find. But before we could start with our work on the model or algorithm, we had to clarify beforehand or speak with our mentor, which kinds of deadwrites should be found because this was not clearly defined on the task-sheet.

3.1 Which kinds of "Dead-Writes" should be now detected by our written Code-Analysis and subsequently removed?

4 Project consists of:

Figure 1: Shows which files our project consists of on "Github".

4.1 First Tests to get familar with Jalangi

```
16  // Level 1 tests
17  var a; // init no read no write
18  var b; b = 3; // init and assign, no write
19  var bb = 3; // init and assign, no write
20  var d; d = 3; d = 4; // init and assign x2, no read
21  var e = function() { console.log("test")};
    // init and function assign, no read/execute
22  var f = {a: "test"} // init and object assign, no read
23  var g = new Object(); g.prototype.a = "test"
    // init and write object, no read
24  var h = []; // init empty array
25  var i = []; i[3] = 6; // init array, write to array, no read
```

(a) Level 1...

```
16  var a = function() { var a; var aa = 3; a = 4};
    // init function and two variables inside the function, no read
17  var b = new Object(); b.prototype.b = function ()
    { var a; var aa = 3; a = 4}; // object assignment via prototypes
```

(b) Level2...

Figure 2: First Tests to get familar with Jalangi

4.2 Our Jalangi-Code-Analysis / Algorithm

4.3 Sample Applications which contains "Dead-Writes"

5 Results

5.1 Problems which occured during our work

Initially it was a big challenge to familiarize our self with Jalangi. There are exists documentations and introductory examples, but from our personal point of view, they are not really easy to understand.

5.2 Conclusion

5.3 Outlook

6 References

-[1]- https://github.com/Samsung/jalangi2
-[2]- http://draw.io/
-[3]- https://www.teamviewer.com/en/
-[4]- https://www.vmware.com
-[5]- https://nodejs.org/en/