# Generating test cases with interesting opportunities for dead code elimination

COLIN ARNET & JULIAN NEFF

## 1 PROJECT PROPOSAL

### 1.1 Introduction

Compiler optimizations are fundamental to improve program performance. It is therefore beneficial to test the capabilities of compiler optimizations. Dead code elimination (DCE) tries to remove code that is never executed or has no effect on the output. This way the programs avoid unnecessary computations. DCE is connected with other optimization techniques like constant propagation. If DCE is successful in removing dead code we can generalise to a certain degree that the compiler performs good optimizations.

In the recent paper by Theodoridis et al. [1] they make use of the connection between DCE and other optimizations. They test DCE by inserting function calls to each basic block. By compiling and executing the programs with different optimization levels we can check if the inserted functions are still in the executable. If they are not, this means that the compiler removed the basic block and therefore successfully performed DCE. By repeating this process with different compilers and optimization flags it is possible to compare the effectiveness of the compiler optimizations and report missing opportunities.

### 1.2 Problem

At the moment the tool introduced in [1] uses randomly generated programs by CSmith [2]. CSmith generates C-programs that do not take inputs and avoid undefined and unspecified behaviors. Because the programs are randomly generated they do not focus on creating tricky DCE situations for the compiler. For this reason we will try to add DCE opportunities to the programs.

### 1.3 Our Approach

One reason for dead code are conditional branches that always resolve to the same result. Our goal is to insert such conditional branches into programs. One way to do this, is to gather all possible values of a variable during the runtime. Then we can insert the following if-condition:

```
if(var == var_new){DCECheck();}
```

Where `var_new` is a value that is never reached by the variable `var`. It is clear that the condition will always be false which results into a dead code block. For our project we will focus on implementing and evaluating this method. For this we will make use of `dead`, the tool introduced by [1]. We will include our approach to modifiy the CSmtih programs to have more challenging DCE opportunities.

### 1.4 Work Schedule

We plan to include our new method to `dead` for the progress report due 29th April. After that we will perform extensive evaluation of our extension and finally write the final report.

## REFERENCES

[1] M. R. Theodoros Theodoridis and Z. Su, "Finding missed optimizations through the lens of dead code elimination," 2022.
[2] E. E. Xuejun Yang, Yang Chen and J. Regehr, "Finding and understanding bugs in c compilers," 2011.

Author's address: Colin Arnet & Julian Neff, carnet@student.ethz.ch,neffj@student.ethz.ch.