Proposal of project: Small code with large binaries

CHRISTOPHER TIBALDO (17-915-778)
MARCO HEINIGER (18-733-824)

1 MOTIVATION & PROBLEM

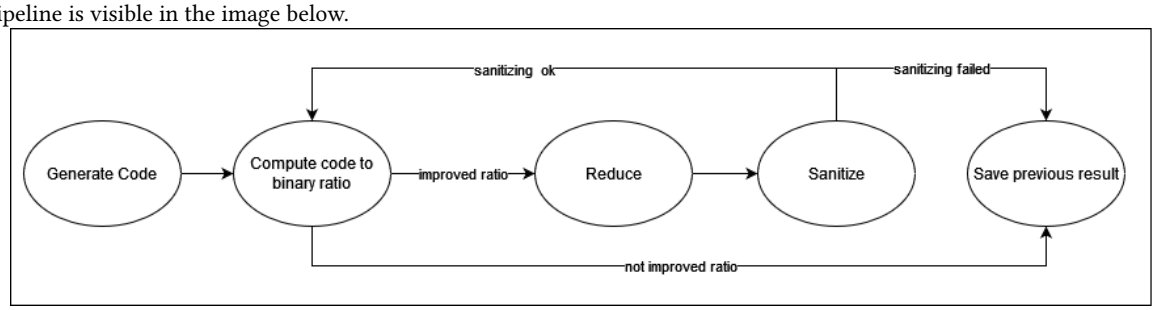
This project aims to investigate how compilers produce large binaries from small pieces of code. The compiler creating these binaries is not necessarily buggy, but code bloat is a good indicator for failed optimizations [2]. Even though our computing and storage capabilities keep expanding, keeping binaries small is still an important factor. The following two examples illustrate why this is the case. Firstly smaller binaries tend to better fit in higher-level caches, thus increasing the speed at which code can be executed. Secondly the proliferation of embedded devices with limited storage capabilities necessitates keeping binaries small [3].

2 RELATED WORK

Reducing the size of binaries has previously been explored through various projects, such as through the use of compiler flags to tune outputs [3]. We will also employ the established approach of "test input reduction" [4], where code gets reduced to a minimum, while still keeping the undesired behaviour. In our case, this behaviour is not the trigger of an error, but an excessive lines-of-code to binary size ratio. We note the existence of benchmarks, measuring code-size reduction performance on real-world code [1]. Since we are looking for worst-case bloat, we won't make use of the constraints these benchmarks impose.

3 APPROACH

The problem of finding large binaries with small code is an exploratory task that can be automated straightforwardly. The approach to tackle our problem is to build a semi-automated pipeline based on several existing libraries. The base language of our project will be python, with which we can create code snippets in the c language. The structure of the pipeline is visible in the image below.



We first either generate code with "csmith" or, time permitting, we feed our pipeline with handwritten code. We then use this code to calculate the code-to-binary ratio. Afterward, we use the library "creduce" which tries to reduce the size of the code, while keeping specific output characteristics unchanged (this might be the presence of specific error messages or in our case binary size). The next step is to use the library "diopter" which checks the code for undefined behaviour or compiler issues. Here we have two possibilities, either "diopter" tells that the code is okay or that it has

Authors' addresses: Christopher Tibaldo (17-915-778), tibaldoc@student.ethz.ch; Marco Heiniger (18-733-824), mheinig@student.ethz.ch.

some issues. If the latter is the case, then the previous version gets saved to the result with the corresponding ratio. If no issue is detected, the ratio of the newly reduced code gets computed. With the computed ratio, our pipeline then decides if the ratio has improved by a threshold or not. If it is the case, the process with "creduce" gets repeated. Else, we save the previous result with the corresponding ratio.

This pipeline is completely feasible to reproduce as all libraries are open-source and written in the same programming language. Other projects focus more on finding which bugs create big binaries, as already stated in the 2 section. In contrast to other work, we will focus on the ratio we get after applying "creduce".

4 WORK SCHEDULE

Our Goals until the 30.04.2023 are the following:

- Getting familiar with the libraries
- A working implementation of our proposed pipeline

Next, we intend to further optimize our pipeline to improve the code-to-binary ratio. After this, we will document our received results in our final report such that we can hand it in by June 06, 2023.

REFERENCES

[1] Anderson Faustino Da Silva, Bruno Conde Kind, José Wesley de Souza Magalhães, Jerônimo Nunes Rocha, Breno Campos Ferreira Guimaraes, and Fernando Magno Quinão Pereira. 2021. Anghabench: A suite with one million compilable c benchmarks for code-size reduction. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 378–390.

[2] Kyriakos Georgiou, Zbigniew Chamski, Andres Amaya Garcia, David May, and Kerstin Eder. 2022. Lost in translation: Exposing hidden compiler optimization opportunities. *Comput. J.* 65, 3 (2022), 718–735.

[3] Masayo Haneda, Peter MW Knijnenburg, and Harry AG Wijshoff. 2006. Code size reduction by compiler tuning. In *Embedded Computer Systems: Architectures, Modeling, and Simulation: 6th International Workshop, SAMOS 2006, Samos, Greece, July 17-20, 2006. Proceedings 6*. Springer, 186–195.

[4] Andreas Zeller. 1999. Yesterday, my program worked. Today, it does not. Why? *ACM SIGSOFT Software engineering notes* 24, 6 (1999), 253–267.