Progress report: Small code with large binaries

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

1 COMPLETED TASKS

As outlined in the project proposal we have started working with the Diopter framework [1]. This framework allows us to run Csmith (Our code generation tool), Creduce (the tool trying to maximize the code-to-binary ratio) as well as a sanitization check on the generated code. In addition to that, we created a Docker image which automatically sets up all required dependencies. This is not strictly necessary for working on a local machine, but it gives us the possibility, if runtimes get unreasonably large, to quickly setup the project on a more powerful remote machine.

When using the Creduce library, we have to design a test function, which the tool can use as a heuristic to evaluate if it is approaching our intended goal. We expect the project to keep following the pattern of designing a new test function by adding more restrictions to the existing ones, until interesting results appear.

Our progress with designing test functions has worked as follows:

(1) Usage of only code ratio as heuristic. The code ratio is calculated as follows $\frac{binaryLength - emptyAssemblySize}{codeLength}$, where the empty assembly is the trivial code "void main(){}". By using this metric, we get very small pieces of code (one or two lines). We think such pieces of code are uninteresting and thus decided to add further constraints.
(2) Setting a minimum amount of lines of code which Creduce should keep when reducing. This has lead to the issue of generating code with many lines that do not contain any meaningful operations (such as lines only containing ";").
(3) We then concluded that to meaningfully count the size of the code, we must be able to count the logical operations performed. To do this, we started parsing the Abstract Syntax Tree (AST) of generated code. By doing this, we can remove uninteresting code (for instance, lines containing only a ";" will appear as "NullStmt" in the AST). The advantage of this method is that it gives us a very fine-grained tool to decide what code to count. The disadvantage is that the reduction process now takes significantly longer.

By having a fully running reduction process, we have achieved the goals we set ourselves in the project proposal for the first part of the work.

2 UPCOMING TASKS

One challenge which we are facing is that Creduce generates code with many unused functions and variables. We believe that by tackling this we will be able to get interesting results. We might also explore how, by differently weighting the parameters in the code ratio calculation, we might generate interesting results.

REFERENCES

[1] 2023. DeadCodeProductions/Diopter. https://github.com/DeadCodeProductions/diopter. [Online; accessed 30-April-2023].

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