Challenges in C Program Repair: Test-Independent and Preprocessor Directives

Anonymous Author(s)

1 INTRODUCTION

As deep learning (DL) has many advantages, such as avoiding the tedious manual feature extraction, DL-based Automated Program Repair (APR) has become a hot topic in software engineering in recent years. The powerful generative capabilities of large language models [1, 2] have opened up new opportunities and brought renewed hope to the field of program repair. Despite the promising results, the program repair still confronts several challenges that must be addressed to enhance the effectiveness and efficiency of the repair process. This paper will delve into two critical challenges in industrial application scenarios: test-independent scenarios and preprocessor directives in C.

2 CHALLENGES

2.1 Challenge one: new APR approaches for test-independent scenarios need to be found.

DL-based APR approaches have exhibited promising performance in evaluation [3], but most of the approaches are evaluated in the test-based scenario: the input program has an associated suite of unit tests with at least a failed one, and the goal is to modify the program such that all the tests pass. The test suite is usually assumed to contain unit tests, which are repeatedly executed and each execution takes relatively a short time. This technique is also utilized in the field of code generation to measure the accuracy of the generated code, such as CodeX [4].

Apart from that, many bugs are not revealed by tests in practice. Large IT companies often use multiple methods to detect bugs [5], such as automatic program analysis tools, interactive program verification, check logs, and manual code review. These bugs do not have an associated test. Furthermore, even if there are tests associated with the bug, executing them may require a complex, distributed environment and take a long time, which is infeasible on the developer's machine. The industrial sector requires software that can operate efficiently and safely, but many test-based APR approaches are not suitable for this purpose. In all the above cases, we cannot assume a test suite for validating the patches. We call such scenario *test-independent*.

There are currently APR techniques that do not rely on test cases, but they are not yet capable of meeting industrial application scenarios. For instance, the R2Fix [6] tool can generate patches based on user defect reports and templates, but it requires detailed English reports and can only fix simple defects. Similarly, LeakFix [7] is another non-testing defect repair tool that can use static analysis techniques to generate patches, but it is limited to repairing specific types of defects. Thus, we need new APR approaches which can validate candidate patches without test case.

2.2 Challenge two: new approaches for handling C preprocessor directives need to be found.

Handling symbols in the codebase is crucial in APR. It can be challenging to correctly handle the symbols in the C code before preprocessing, as user-defined preprocessing directives may disrupt the syntactic structure of the source code.

In existing APR studies in C, it is often assumed that the input program is already pre-processed and the patch is applied to the pre-processed program [8]. However, we found that in real industrial settings this assumption may not hold. To obtain the pre-processed program, it is essential to have access to the entire source code. Nevertheless, it may be difficult to obtain the complete source code due to company's security policy or compliance restriction.

To correctly handle the symbols in the code before preprocessing, the following difficulties will be encountered in practice. First, the project may be compiled with complex build directives with user-defined program transformations, and it is difficult to either automatically extract symbols from buggy files or ask the end user to extract the pre-processed program.

Another possibility is to treat the pre-process directives as undefined variables or procedures, but due to the complexity of programs, this approach often fails. Therefore, it is imperative to find new approaches for handling C preprocessor directives.

REFERENCES

[1] Niklas Muennighoff et al. 2022. Crosslingual generalization through multitask finetuning. (2022). arXiv: 2211.01786 [cs.CL].

[2] Wang Haining. 2022. Development of natural language processing technology. *ZTE Communications*, 28, 02, 59–64.

[3] Qihao Zhu, Zeyu Sun, Yuan-an Xiao, Wenjie Zhang, Kang Yuan, Yingfei Xiong, and Lu Zhang. 2021. A syntax-guided edit decoder for neural program repair. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 341–353.

[4] Mark Chen et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

[5] Jing Han, Tong Jia, Yifan Wu, Chuanjia HOU, and Ying LI. 2021. Feedback-aware anomaly detection through logs aware anomaly detection through logs for large for large-scale software systems scale software systems. *ZTE Communications*, 19, 88–94.

[6] Chen Liu, Jinqiu Yang, Lin Tan, and Munawar Hafiz. 2013. R2fix: automatically generating bug fixes from bug reports. In *2013 IEEE Sixth international conference on software testing, verification and validation*. IEEE, 282–291.

[7] Qing Gao, Yingfei Xiong, Yaqing Mi, Lu Zhang, Weikun Yang, Zhaoping Zhou, Bing Xie, and Hong Mei. 2015. Safe memory-leak fixing for c programs. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1. IEEE, 459–470.

[8] Yufeng Cheng, Meng Wang, Yingfei Xiong, Zhengkai Wu, Yiming Wu, and Lu Zhang. 2017. Un-preprocessing: extended CPP that works with your tools. In *Internetware*. Hong Mei, Jian Lyu, Zhi Jin, and Wenyun Zhao, (Eds.), 3:1–3:10.