Program Repair Competition

Ridwan Shariffdeen
National University of Singapore
ridwan@comp.nus.edu.sg

Martin Mirchev
National University of Singapore
mmirchev@comp.nus.edu.sg

Abhik Roychoudhury
National University of Singapore
abhik@comp.nus.edu.sg

***Abstract*—Automated Program Repair(APR) is a rapidly developing technology that can assist developers in fixing software system errors. An essential part of any scientific work is a fair and comprehensive evaluation of the state-of-the-art techniques. At present there is no publicly available standard mechanism to perform comparisons for program repair. In this paper, we propose such a standardization in the form of a competition to foster comparability, advancing the state-of-the-art capabilities and to pave the path for the next-generation repair technology.**

***Index Terms*—automated program repair, competition**

I. Introduction

Improving the productivity of software developers by relieving the burden of manually fixing the ever-increasing amount of software bugs has been the goal of automated program repair technology [1]. The past decade has seen a significant growth in Automated Program Repair (APR) expanding into various tasks in Software Engineering including but not limited to fixing functional errors, repairing security vulnerabilities, assisting educators in teaching programming languages and collateral evolution of software systems [2].

In general, the repair problem can be formulated as a search problem where; given a buggy program `P` and a program specification σ (i.e. a test-suite) which `P` does not satisfy, find an edit to the program such that it satisfies σ. In practice, σ can be represented in various different forms. For example, the program specification σ could be presented as a set of test-cases or as a program assertion that should hold true for all inputs of the program. The property being specified by the specification could capture correct functionality of the program or a security property. Repair techniques [2] proposed in the literature assume different types and formats to the inputs albeit attempting the same repair task.

A recent study [3] showed appreciable interest from the software practitioners to incorporate repair technology in their development processes, provided the tools can be integrated in a non-disruptive manner. In order for a repair tool to be used it is important that the existing repair tools improve its usability such that users can easily operate the tools and is provided with documentation support on how to configure/extend its capabilities. The benefits of developing the tools beyond an academic prototype has little-short term reward but longer-term benefits for wide adoption of repair technology. However, such considerations are generally not included when developing repair techniques, creating an entry barrier for developers to familiarize with the repair technology.

Furthermore, there is no widely adopted set of benchmark programs that remains relevant with the development of the field. Most of the benchmark subjects quickly become obsolete either due to the use of deprecated dependent libraries or simply because the code is no longer maintained. Such programs make it difficult to use to assess the advancement made in the technology and irrelevant for state of the art comparisons. Using out-dated subjects as benchmark programs makes it challenging for researchers to compare the performance of their techniques, with no reward for the effort. It is also important for the standardized benchmark programs to evolve over time in order to prevent repair techniques from being over-fitting for a given set of benchmark programs.

Finally, the recent emergence of large language models (LLMs) [4] has the potential to automatically generate code and will change the traditional processes in software engineering. Traditional program repair technology mainly focused on fixing human-errors in programs by learning fix-patterns from fixes written by developers themselves. In future, software may comprise of code generated by humans and machines, hence it is important as a community to develop techniques that can *co-exist* with AI-driven Software Development. Thus, we can explore how automatic program repair can deliver last-mile improvement to automatically generated code from LLMs with the vision of trusted automated programming.

We propose a *Program Repair Competition* that provides additional incentive to the community to setup and maintain repair tools and benchmarks relevant for the current needs in practice. A competition can foster the development of standards, tools and benchmarks that can be used to assess the advancement of the technology. It can also provide reward for technical improvements while providing a platform for researchers to compare with state-of-the-art techniques. A competition could also enforce requirements / standards for repair tools to agree on. Such a standard can specify how the input and output for the repair problem will be specified, the qualitative / quantitative measures used to compare techniques and the types of different program specifications that can be repaired. These standards should make it easy to generate and compare new repair techniques. A common platform with tool-agnostic, user-friendly interfaces allows to even add new software programs to be repaired with dynamic workflows.

II. Methodology

The *program repair competition* will be designed to evaluate state-of-the-art repair tools with respect to effectiveness, efficiency and quality of the results. Effectiveness will be measured using its repair capabilities on different repair tasks while efficiency is measured with respect to usability aspect (i.e. time latency, resource usage) of the tool. Quality of the results (i.e. generated patches) will be evaluated. The main *goals* of the competition are:

1) mechanism to generate data for the advancement of program repair technology
2) provide incentives for researchers to improve their techniques beyond academic prototypes;
3) provide visibility for the state-of-the-art repair tools to the software practitioners;
4) establish an evolving community standard to evaluate new repair techniques (i.e. quality vs quantity);

Organization: : The competition is designed in a manner inspired by the successful model used in SV-COMP [5], the International Competition on Software Verification. The organizing committee would comprise of a 'core' team and a 'jury' committee. The 'core' team would be responsible for conducting the evaluations in an automated, reproducible and fair manner. The jury committee is formed with a representative of each of the participating teams. The jury committee would serve as an advisory board for decision making on qualification of tools, benchmarks and setting up standards.

Timeline: The competition will be announced at the International Workshop on Automated Program Repair co-hosted with International Conference on Software Engineering (ICSE) 2023. The tool developers will submit their intention to participate in the competition and provide an initial version of their tool which can be used in pre-runs by the organizing committee and report preliminary results to the jury committee and the tool developers. Once the jury committee approves the final list of benchmarks for evaluation, the tool developers will submit the final versions which will be independently evaluated by the organizing committee. Final results will be reported to the team for inspection and approval, and the results will be announced on the competition website. The competition organizers write the final competition report with reviews / approval from the jury committee. These tentative plans will be firmed up during the process.

Standards: The organizing committee will announce the rules and regulations for the competition along with the metrics used to evaluate the repair tools. The rules will define the minimum qualification for a repair tool to participate in the competition, the standard interfaces it should implement and the expected inputs for the repair task. In addition, a clear definition of the categories and sub-categories used to determine different capabilities of repair will be announced to the participants. Finally, the scoring criteria will be approved by the running jury committee before the evaluation.

Competition Tracks: : Several tracks run in parallel capturing different repair tasks including but not limited to fixing logical-errors, vulnerability repair, last-mile fixes on novice programs and auto-generated code, fixing reports generated by static code analysers and generating feedback for student programming assignments. Each track consists of repair tasks from multiple different languages extracted from real-world applications. Each iteration of the competition will update the tasks in each track to include new challenges in order to circumvent the risk of tools that over-fit.

Evaluation: : The scoring criteria should be designed to identify techniques that generate high-quality patches to be ranked higher, while detecting tools that generate undesirable outputs to be ranked lower. Using an approved scoring criteria by the running jury committee preliminary results would be provided to the participating teams. The final evaluation is an adversarial competition between participating teams. Competing teams may provide proof of incorrectness for the patches generated by a team, to dispute their scores.

III. Conclusion

Automated program repair can improve the trustworthiness of both manually written and automatically generated code. Due to the rapid advancements in repair technology and the fast-changing landscape in software engineering there is a need to setup a continuously improving set of standards. Inspired by the success in the fields of software verification and testing, we propose a *Program Repair Competition*. At the end of the evaluation all repair tasks, results and tools will be published for reproducibility purposes. Infrastructure, data and all components will be made available for public access.

The potential benefits of participating includes high-visibility of the technology among software practitioners, acknowledgement for technical improvements and advancing the state of the art research in program repair. We invite the community to participate in this program repair competition, which will hopefully play a significant role in shaping the next-generation of repair technology.

Acknowledgment

The work is partially supported by a Singapore Ministry of Education (MoE) Tier 3 grant "Automated Program Repair", MOE-MOET32021-0001.

References

[1] C. Le Goues, M. Pradel, and A. Roychoudhury, "Automated program repair," *Commun. ACM*, vol. 62, no. 12, p. 56–65, nov 2019. [Online]. Available: https://doi.org/10.1145/3318162

[2] M. Monperrus, "The Living Review on Automated Program Repair," HAL Archives Ouvertes, Technical Report hal-01956501, 2018. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01956501

[3] Y. Noller, R. Shariffdeen, X. Gao, and A. Roychoudhury, "Trust enhancement issues in program repair," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2228–2240. [Online]. Available: https://doi.org/10.1145/3510003.3510040

[4] T. B. Brown *et al.*, "Language models are few-shot learners," *arxiv:2005.14165*, 2020.

[5] D. Beyer, "Software verification: 10th comparative evaluation (sv-comp 2021)," in *Tools and Algorithms for the Construction and Analysis of Systems*, J. F. Groote and K. G. Larsen, Eds. Cham: Springer International Publishing, 2021, pp. 401–422.