An extended analysis of the cause of software regressions in Python and JavaScript

Feng Dai, Yuefeng Hu, Tetsuro Yamazaki, Shigeru Chiba

The University of Tokyo

What is Software Regression?

- A software bug: a feature that has worked before stops working ***due to changes of the code***
- Most tools try to find regressions instead of finding the causes of regressions

Existing technique

- Hu et al., SLE 2024
- Runtime tracing of inputs and outputs
- Based on JavaScript dataset
- Limitations
  - Accuracy not satisfying, only for JavaScript

We want to extend this work..

- To improve accuracy
  - Proposing 2 new heuristics, and 1 LLM-powered technique
- To support another language, Python
  - Organizing a Python dataset
  - Implementing a Python tracer

Evaluation

Dataset	FDF	Top-2	First code change	Deepest code change	LLM-powered technique
Python	6	8	7	9	9
JS	6	3	6	8	9
Total	12	11	13	17	18

12 regressions from Python dataset and 11 regressions from JavaScript dataset:

- FDF algorithm and Top-2 algorithm are from Hu et al. 2024
- Both our heuristics outperform the algorithms from Hu et al.
- The LLM-powered technique identifies the most causes of regressions in both Python and JavaScript datasets
- Failed cases are affected by unrelated information in source code, such as random variables, and big code refactoring

Conclusion

- Instead of tracing inputs and outputs, tracing the order of function executions and code changes has better accuracy
- An LLM-powered technique combined with error messages of regressions achieves the best accuracy

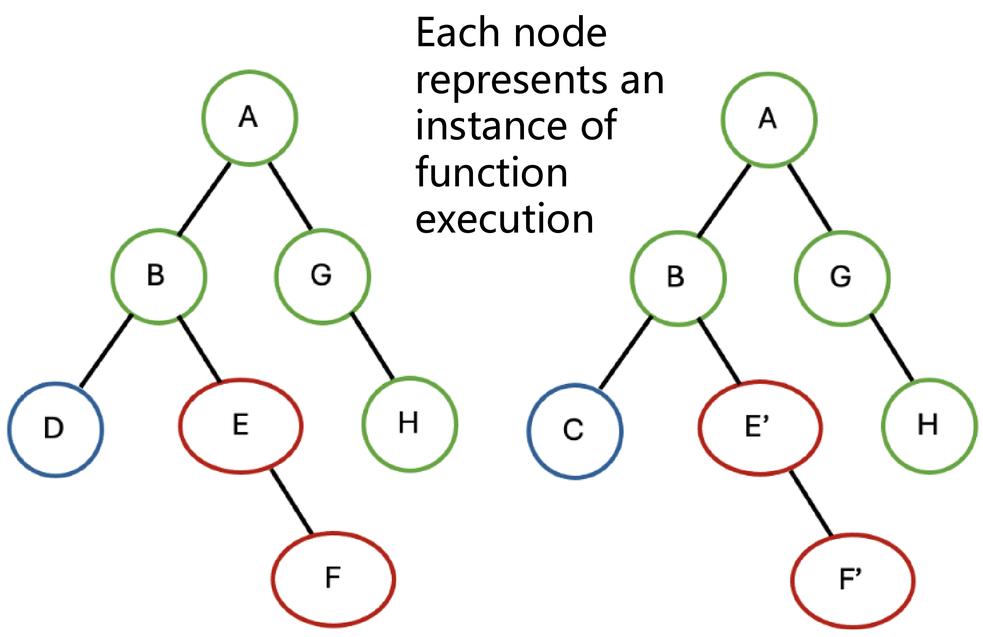
Reference

[1] Hiromu Ishibe. 2023. A cause detector of software regressions by comparing program execution traces. https://csg-www.s3.ap-northeast-1.amazonaws.com/public/papers/23/master-ishibe.pdf

[2] Yuefeng Hu, Hiromu Ishibe, Feng Dai, Tetsuro Yamazaki, and Shigeru Chiba. 2024. Bugfox: A Trace-Based Analyzer for Localizing the Cause of Software Regression in JavaScript. In Proceedings of the 17th ACM SIGPLAN International Conference on Software Language Engineering (SLE '24). Association for Computing Machinery, New York, NY, USA, 224–233. https://doi.org/10.1145/3687997.3695648

Idea: trace source code change

I. Transform both the base and faulty program into **tree structures** where each node represents one function execution



(a) The base version. **(b)** The faulty version.

II. Trace only the order of function executions and source code changes instead of input/output changes

Heuristic 1: Find the first change of source code at the same position of two function executions

- **The same position** means two nodes have the same depth and breadth, and also the same name and the same type
- In the example, E and E' have the same name but different function body, and they are at the same position
- E and E' are the first pair of code change, and are reported by this heuristic

Heuristic 2: Find the deepest change of source code at the same position of two function executions

- In the example, F and F' are the deepest pair of code change, and are reported by this heuristic

LLM-powered technique: Linearize function executions, and let an LLM compare two function executions

- To avoid input limitation, we only part of the function execution
- We also give an LLM the error messages of the regression