Can we quantify bottlenecks in Symbolic Execution?

Mahyar Emami, Rishabh Iyer

ABSTRACT

Symbolic execution (SE) is an automated program analysis technique that has widespread use in tools for bug finding [1, 2, 4], formal verification [7, 8, 10] and performance analysis [5, 6, 9]. However, despite its widespread use, the performance of symbolic execution (i.e., how fast it can analyze the target program) remains a concern with the current consensus being that it can never really scale to exhaustively analyze codebases with millions of lines of code [3]. Despite this concern, there exists no concrete understanding of the major bottlenecks in SE and how each of them affect performance. We believe that this is due to the fact that SE performance is a complex function of the specificities of the target program and the particular implementation of the SE tool. In this work, we aim to shed light on this complex function and make quantitative statements about existing bottlenecks in SE performance, in order to enable targetted optimizations that can allow SE to scale to larger codebases.

Concretely, the performance metric is the time taken by the SE tool to achieve a certain code coverage on the target program. The factors that affect performance may be one or more of the structure of the target program, its instruction mix, the amount of state stored, the type of SMT queries generated when analyzing it, the implementation of the SE tool, etc. We expect identifying the precise dependence of SE performance on each of these factors to be a complex undertaking.

1. REFERENCES

[1] C. Cadar, D. Dunbar, and D. R. Engler. KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings*, 2008.

[2] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. EXE: automatically generating inputs of death. *ACM Trans. Inf. Syst. Secur.*, 2008.

[3] C. Cadar and K. Sen. Symbolic execution for software testing: three decades later. *Commun. ACM*, 56(2):82–90, 2013.

[4] P. Godefroid, M. Y. Levin, and D. A. Molnar. SAGE: whitebox fuzzing for security testing. *Commun. ACM*, 2012.

[5] Y. Hu, G. Huang, and P. Huang. Automated reasoning and detection of specious configuration in large systems with symbolic execution. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, 2020.

[6] R. R. Iyer, L. Pedrosa, A. Zaostrovnykh, S. Pirelli, K. J. Argyraki, and G. Candea. Performance contracts for software network functions. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, 2019.

[7] L. Nelson, J. V. Geffen, E. Torlak, and X. Wang. Specification and verification in the field: Applying formal methods to BPF just-in-time compilers in the linux kernel. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, 2020.

[8] L. Nelson, H. Sigurbjarnarson, K. Zhang, D. Johnson, J. Bornholt, E. Torlak, and X. Wang. Hyperkernel: Push-button verification of an OS kernel. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, 2017.

[9] L. Pedrosa, R. R. Iyer, A. Zaostrovnykh, J. Fietz, and K. J. Argyraki. Automated synthesis of adversarial workloads for network functions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018*, 2018.

[10] A. Zaostrovnykh, S. Pirelli, R. R. Iyer, M. Rizzo, L. Pedrosa, K. J. Argyraki, and G. Candea. Verifying software network functions with no verification expertise. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, 2019.