BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

https://fit.vut.cz

Project Practice 2019/2020

Static Analysis in Facebook Infer Focused on Atomicity

Dominik Harmim*

Abstract
The goal of this project practice is to improve, extend, and perform new experiments with *Atomer* – a *static analyser* that detects *atomicity violations*. Atomer was proposed and implemented within a bachelor's thesis at FIT BUT as a module of *Facebook Infer*, which is an open-source and extendable static analysis framework that promotes efficient *modular* and *incremental* analysis. The original analyser works on the level of *sequences of function calls*, but in this project, it was changed to work on the level of *sets of function calls*. The solution is based on the assumption that sequences executed *atomically once* should probably be executed *always atomically*. Within this project, two new main features were introduced: (i) support for *C++ and Java locks*, (ii) and distinguish *multiple locks used*. The new features were successfully verified and evaluated on smaller programs created for testing purposes. Furthermore, new experiments on *publicly available real-life extensive programs* were made.

Keywords: Facebook Infer — Static Analysis — Abstract Interpretation — Contracts for Concurrency — Atomicity Violation — Concurrent Programs — Program Analysis — Atomic Sets — Atomicity — Incremental Analysis — Modular Analysis — Compositional Analysis — Interprocedural Analysis

Supplementary Material: Atomer Repository — Atomer Wiki — VeriFIT Static Analysis Plugins — Facebook Infer Repository

*xharmi00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Bugs are an integral part of computer programs ever since the inception of the programming discipline. Unfortunately, they are often hidden in unexpected places, and they can lead to unexpected behaviour, which may cause significant damage. Nowadays, developers have many possibilities of catching bugs in the early development process. *Dynamic analysers* or tools for *automated testing* are often used, and they are satisfactory in many cases. Nevertheless, they can still leave too many bugs undetected, because they can analyse only particular program flows dependent on the input data. An alternative solution is *static analysis* that has its shortcomings as well, such as the *scalability* on large codebases or considerably high rate of incorrectly reported errors (so-called *false positives* or *false alarms*).

Recently, Facebook introduced *Facebook Infer*: a tool for creating *highly scalable*, *compositional*, *incremental*, and *interprocedural* static analysers. Facebook Infer has grown considerably, but it is still under active development by many teams across the globe. It

is employed every day not only in Facebook itself, but also in other companies, such as Spotify, Uber, Mozilla, or Amazon. Currently, Facebook Infer provides several analysers that check for various types of bugs, such as buffer overflows, data races and some forms of deadlocks and starvation, null-dereferencing, or memory leaks. However, most importantly, Facebook Infer is a framework for building new analysers quickly and easily. Unfortunately, the current version of Facebook Infer still lacks better support for *concurrency* bugs. While it provides a reasonably advanced *data race* analyser, it is limited to Java and C++ programs only and fails for C programs, which use a more *low-level* lock manipulation.

In *concurrent programs*, there are often *atomicity requirements* for execution of specific sequences of instructions. Violating these requirements may cause many kinds of problems, such as unexpected behaviour, exceptions, segmentation faults, or other failures. *Atomicity violations* are usually not verified by compilers, unlike syntactic or some sorts of semantic rules. Moreover, atomicity requirements, in most cases, are not even documented at all. So in the end, programmers themselves must abide by these requirements and usually lack any tool support. Furthermore, in general, it is difficult to avoid errors in *atomicity-dependent programs*, especially in large projects, and even more laborious and time-consuming is finding and fixing them.

In the thesis [12], there was proposed the *Atomer* – a static analyser for finding some forms of atomicity violations implemented as a module of Facebook Infer. In particular, the stress is put on the *atomic execution of sequences of function calls*, which is often required, e.g., when using specific library calls. The idea of checking atomicity of certain sequences of function calls is inspired by the work of *contracts for concurrency* [10]. In the terminology of [10], atomicity of specific sequences of calls is the most straightforward (yet very useful in practice) kind of contracts for concurrency. The implementation mainly targets C/C++ programs that use *PThread* locks. Within this project practice, Atomer was improved, extended, and other experiments were performed. In particular, two new main features were introduced: (i) support for *C++ and Java locks*, (ii) and distinguish *multiple locks used*. Moreover, working with *sequences of function calls* was approximated by working with *sets of function calls* to make the solution more scalable.

The development of Atomer has been discussed with developers of Facebook Infer, and it is a part of the H2020 ECSEL project Aquas. Parts of this paper are taken from the thesis [12] and the paper [13] written in collaboration with Vladimír Marcin and Ondřej Pavela.

The rest of the paper is organised as follows. In Section 2, there is described Facebook Infer framework. Atomer is described in Section 3, together with all the extensions and improvements implemented within this project practice. Subsequently, Section 4 discusses the experimental evaluation of the new features and other experiments that were performed in this project. Finally, Section 5 concludes the paper.

2. Facebook Infer

[[foo]]

3. Atomicity Violations Detector

[[foo]]

4. Experimental Evaluation

[[foo]]

[4] [10] [2] [8] [21] [22] [13] [19] [24] [15] [16] [20] [7] [6] [5] [9] [25] [11] [3] [18] [26] [23] [1] [17] [14]

5. Conclusion

This paper started by discussing a *static analysis* framework that uses *abstract interpretation* — *Facebook Infer* — its features, architecture, and existing analysers implemented in this tool. The major part of the paper then aimed at the description of a static analyser for detecting *atomicity violations* — *Atomer* — implemented as a module of Facebook Infer and its extensions and improvements. Lastly, it is described the experimental evaluation of the new features and other experiments performed in this project practice, and it is discussed possible future work.

The original analyser works on the level of *sequences of function calls*, but in this project, it was changed to work on the level of *sets of function calls*. The solution is based on the assumption that sequences executed *atomically once* should probably be executed *always atomically*. Within this project, two new main features were introduced: (i) support for *C++ and Java locks*, (ii) and distinguish *multiple locks used*.

The introduced extensions and improvements were successfully tested on smaller *hand-crafted* programs. It turned out that such innovations enhanced the *accuracy* and *scalability* of the analysis. Moreover, Atomer was experimentally evaluated on another software. Notably, it was evaluated on *open-source real-life Java programs* – *Apache Cassandra* and *Tomcat*. Already

fixed and reported real bugs were successfully rediscovered. Nevertheless, so far, quite some *false alarms* are reported. However, a result of the analyser can be used as an input for *dynamic analysis* which can determine whether the reported atomicity violations are real errors.

Atomer again shows the potential for further improvements. The future work will focus mainly on increasing the accuracy of the methods used by, e.g., distinguishing the *context of called functions* by considering *formal parameters*, *ranking* of atomic functions, or focusing on *library containers concurrency restrictions* related to method calls. Further, it is needed to perform more experiments on *real-life* programs with an effort to find and report *new bugs*.

The code of Atomer is available on GitHub as an *open-source repository*. The *Pull Request* to the `master` branch of Facebook Infer's repository is currently the work under progress. It is expected that work on this project will continue not only within diploma thesis at FIT BUT.

Acknowledgements

I want to thank my supervisor Tomáš Vojnar for his assistance. Further, I would like to thank other colleagues from VeriFIT. I would also like to thank Nikos Gorogiannis from the Infer team at Facebook for useful discussions about the development of the analyser. Lastly, I thank for the support received from the H2020 ECSEL project Aquas.

References

[1] ALLEN, F. E.: Control Flow Analysis. In: *Proceedings of a Symposium on Compiler Optimization*. Urbana-Champaign, Illinois: ACM, New York, NY, USA, 1970, p. 1 – 19. doi:10.1145/800028.808479.

[2] BLACKSHEAR, S.; GOROGIANNIS, N.; O'HEARN, P. W.; SERGEY, I.: RacerD: Compositional Static Race Detection. *Proceedings of ACM Programming Languages*. October 2018, vol. 2, OOPSLA'18, p. 144:1 – 144:28. ISSN 2475-1421. doi:10.1145/3276514.

[3] BLACKSHEAR, S.; O'HEARN, P. W.: *Open-sourcing RacerD: Fast static race detection at scale* [online]. 2017-10-19 [cit. 2020-07-28]. Available at: https://code.fb.com/android/open-sourcing-racerd-fast-static-race-detection-at-scale.

[4] CALCAGNO, C.; DISTEFANO, D.; O'HEARN, P. W.; YANG, H.: Compositional Shape Analysis by Means of Bi-abduction. In: *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Savannah, GA, USA: ACM, New York, NY, USA, January 2009, p. 289 – 300. POPL'09. ISBN 978-1-60558-379-2. doi:10.1145/1480881.1480917.

[5] COUSOT, P.: *Abstract Interpretation* [online]. 2008-08-05 [cit. 2020-07-28]. Available at: https://www.di.ens.fr/~cousot/AI.

[6] COUSOT, P.: *Abstract Interpretation in a Nutshell* [online]. [cit. 2020-07-28]. Available at: https://www.di.ens.fr/~cousot/AI/IntroAbsInt.html.

[7] COUSOT, P.: Abstract Interpretation Based Formal Methods and Future Challenges, invited paper. In: WILHELM, R., ed.: *« Informatics — 10 Years Back, 10 Years Ahead »*. Springer-Verlag, March 2001, p. 138 – 156. Lecture Notes in Computer Science, vol. 2000. doi:10.1007/3-540-44577-3_10.

[8] COUSOT, P.; COUSOT, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Los Angeles, California: ACM Press, New York, NY, 1977, p. 238 – 252. POPL'77. doi:10.1145/512950.512973.

[9] COUSOT, P.; COUSOT, R.: Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation, invited paper. In: BRUYNOOGHE, M.; WIRSING, M., ed.: *Proceedings of the International Workshop Programming Language Implementation and Logic Programming*. PLILP'92. Springer-Verlag, Berlin, Germany, January 1992, p. 269 – 295. Leuven, Belgium, 13 – 17 August 1992, Lecture Notes in Computer Science 631. doi:10.1007/3-540-55844-6_101.

[10] DIAS, R. J.; FERREIRA, C.; FIEDOR, J.; LOURENÇO, J. M.; SMRČKA, A.; SOUSA, D. G.; VOJNAR, T.: Verifying Concurrent Programs Using Contracts. In: *2017 IEEE International Conference on Software Testing, Verification and Validation*. Tokyo, Japan: IEEE, March 2017, p. 196 – 206. ICST'17. ISBN 9781509060313. doi:10.1109/ICST.2017.25.

[11] GOROGIANNIS, N.; O'HEARN, P. W.; SERGEY, I.: A True Positives Theorem for a Static Race Detector. *Proceedings of ACM Programming Languages*. January 2019, vol. 3, POPL'19, p. 57:1 – 57:29. ISSN 2475-1421. doi:10.1145/3290370.

[12] HARMIM, D.: *Static Analysis Using Facebook Infer to Find Atomicity Violations*. Brno, 2019. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Tomáš Vojnar, Ph.D.

[13] HARMIM, D.; MARIN, V.; PAVELA, O.: Scalable Static Analysis Using Facebook Infer. In: *Excel@FIT*. Brno University of Technology, Faculty of Information Technology, 2019.

[14] KROENING, D.; POETZL, D.; SCHRAMMEL, P.; WACHTER, B.: Sound static deadlock analysis for C/Pthreads. In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. Singapore, Singapore: ACM, New York, NY, USA, September 2016, p. 379 – 390. ASE'16. ISBN 978-1-4503-3845-5. doi:10.1145/2970276.2970309.

[15] LENGÁL, O.; VOJNAR, T.: Abstract Interpretation. Lecture Notes in Formal Analysis and Verification. In:. Brno University of Technology, Faculty of Information Technology, 2018.

[16] MARCIN, V.: *Static Analysis of Concurrency Problems in the Facebook Infer Tool*. Brno, 2018. Project practice. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Tomáš Vojnar, Ph.D.

[17] MEYER, B.: Applying "Design by Contract". *Computer*. October 1992, vol. 25, no. 10, p. 40 – 51. ISSN 0018-9162. doi:10.1109/2.161279.

[18] MINSKY, Y.; MADHAVAPEDDY, A.; HICKEY, J.: *Real world OCaml*. 1st ed. Sebastopol, CA: O'Reilly Media, 2013. ISBN 144932391X.

[19] MØLLER, A.; SCHWARTZBACH, I. M.: *Static Program Analysis*. Department of Computer Science, Aarhus University, October 2018.

[20] NIELSON, F.; NIELSON, R. H.; HANKIN, C.: *Principles of Program Analysis*. Berlin: Springer-Verlag, 2005. ISBN 3-540-65410-0.

[21] REPS, T.; HORWITZ, S.; SAGIV, M.: Precise Interprocedural Dataflow Analysis via Graph Reachability. In: *Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, New York, NY, USA, 1995, p. 49 – 61. POPL'95. ISBN 0-89791-692-1. doi:10.1145/199448.199462.

[22] SHARIR, M.; PNUELI, A.: Two approaches to interprocedural data flow analysis. In: MUCHNICK, S. S.; JONES, N. D., ed.: *Program Flow Analysis: Theory and Applications*. Prentice Hall Professional Technical Reference, 1981, chap. 7, p. 189 – 211. ISBN 0137296819.

[23] SOUSA, D. G.; DIAS, R. J.; FERREIRA, C.; LOURENÇO, J. M.: Preventing Atomicity Violations with Contracts. *CoRR*. 2015, abs/1505.02951. 1505.02951.

[24] VOJNAR, T.: Different Approaches to Formal Verification and Analysis. Lecture Notes in Formal Analysis and Verification. In:. Brno University of Technology, Faculty of Information Technology, 2018.

[25] VOJNAR, T.: Lattices and Fixpoints for Symbolic Model Checking. Lecture Notes in Formal Analysis and Verification. In:. Brno University of Technology, Faculty of Information Technology, 2018.

[26] YI, K.: *Inferbo: Infer-based buffer overrun analyzer* [online]. 2017-02-06 [cit. 2020-07-28]. Available at: https://research.fb.com/inferbo-infer-based-buffer-overrun-analyzer.