Qilin: A New Framework for Supporting Fine-Grained Context-Sensitivity in Java Pointer Analysis (Artifact)

Dongjie He
The University of New South Wales, Sydney, Australia

Jingbo Lu
The University of New South Wales, Sydney, Australia

Jingling Xue
The University of New South Wales, Sydney, Australia

Abstract

Existing whole-program context-sensitive pointer analysis frameworks for Java, which were open-sourced over one decade ago, were designed and implemented to support only method-level context-sensitivity (where all the variables/objects in a method are qualified by a common context abstraction representing a context under which the method is analyzed). We introduce Qilin as a generalized (modern) alternative, which will be open-sourced soon on GitHub, to support the current research trend on exploring **fine-grained context-sensitivity** (including variable-level context-sensitivity where different variables/objects in a method can be analyzed under different context abstractions at the variable level), **precisely**, **efficiently**, and **modularly**. To meet these **four** design goals, Qilin is developed as an imperative framework (implemented in Java) consisting of a fine-grained pointer analysis kernel with parameterized context-sensitivity that supports on-the-fly call graph construction and exception analysis, solved iteratively based on a new carefully-crafted incremental worklist-based constraint solver, on top of its handlers for complex Java features.

We have evaluated Qilin extensively using a set of 12 representative Java programs (popularly used in the literature). For method-level context-sensitive analyses, we compare Qilin with Doop (a declarative framework that defines the state-of-the-art), Qilin yields logically the same precision but more efficiently (e.g., 2.4x faster for four typical baselines considered, on average). For fine-grained context-sensitive analyses (which are not currently supported by open-source Java pointer analysis frameworks such as Doop), we show that Qilin allows seven recent approaches to be instantiated effectively in our parameterized framework, requiring additionally only an average of 50 LOC each.

2012 ACM Subject Classification Theory of computation → Program analysis

Keywords and phrases Pointer Analysis, Fine-Grained Context Sensitivity

Digital Object Identifier 10.4230/DARTS.8.2.6

Funding Supported by ARC Grants DP180104069 and DP210102409.

Acknowledgements We thank all the reviewers for their constructive comments.

Related Article Dongjie He, Jingbo Lu, and Jingling Xue, "Qilin: A New Framework For Supporting Fine-Grained Context-Sensitivity in Java Pointer Analysis", in 36th European Conference on Object-Oriented Programming (ECOOP 2022), LIPIcs, Vol. 222, pp. 30:1–30:29, 2022.
https://doi.org/10.4230/LIPIcs.ECOOP.2022.30

Related Conference 36th European Conference on Object-Oriented Programming (ECOOP 2022), June 6–10, 2022, Berlin, Germany

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2022 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

This artifact contains the binary form of Qilin, together with the source code of Doop [3] (expressed in the form of Datalog rules with a number of bug fixes from us) and a set of 12 Java benchmarks used in our evaluation. The source code of Qilin has been released and maintained at `https://github.com/QiLinPTA/QiLin`.

The artifact can be used to reproduce all the tables and raw data that appear in the evaluation part of our paper. It supports the following four claims that we make in the paper: (1) Qilin delivers exactly the same precision as Doop (the state-of-the-art) for a few commonly used pointer analysis like Andersen's analysis [1], kCFA [5], and kOBJ [4]; (2) Qilin (currently runs in a single thread) outperforms Doop (runs in its best setting, i.e., 8 threads) substantially, with an average speedup of 2.4x; (3) Qilin is very effective in supporting fine-grained context-sensitive pointer analyses; and (4) Qilin is modular in allowing its common codebase to be shared by a wide range of existing pointer analysis techniques.

2 Content

The artifact package includes:

- a Docker image, which contains
  - an executable jar file, i.e., `artifact/pta/Qilin-1.0-SNAPSHOT.jar`,
  - benchmarks (including 9 benchmarks from DaCapo2006 [2] and 3 Java applications),
  - a Java library (i.e., `artifact/pta/lib/jre/jre1.6.0_45`),
  - the scripts for running all experiments and extracting results, and
  - Doop (version 4.24.0),
- a `README.md` file,
- the PDF file of the paper,
- the PDF of the artifact manual, and
- a license file.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: `https://doi.org/10.5281/zenodo.5763519`. Again, we have released Qilin as an open-source tool at `https://github.com/QiLinPTA/QiLin`.

4 Tested platforms

We have carried out all the experiments on an eight-core Intel(R) Xeon(R) CPU E5-2637 3.5GHz machine with 512GB of RAM. The underlying operating system is Ubuntu 20.04. The time budget used for running each pointer analysis on a program is set as 24 hours.

We would like to warn users that **different machines used for running our artifact may result in different speedups and scalability**. Unfortunately, we have thus lost a badge during the artifact evaluation as the reviewers failed to reproduce the results for some large benchmarks (e.g., `eclipse`, `checkstyle`, `findbugs`) due to the memory size differences.

In addition, we have provided detailed documentation on Qilin's wiki page, `https://github.com/QilinPTA/Qilin/wiki`, for describing how to use Qilin as either a library or as a command-line tool and how to write your own analyses in Qilin.

5 License

The artifact is available under license GPL v3.

6 MD5 sum of the artifact

d0524c71bb102a192eb7f3e226d1d446

7 Size of the artifact

1.6 GiB

References

1 Lars Ole Andersen. *Program analysis and specialization for the C programming language*. PhD thesis, University of Cophenhagen, 1994.

2 Stephen M. Blackburn, Robin Garner, Chris Hoffmann, Asjad M. Khang, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. The DaCapobenchmarks: Java benchmarking development and analysis. In *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 169–190, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1167515.1167488.

3 Martin Bravenboer and Yannis Smaragdakis. Strictly declarative specification of sophisticated points-to analyses. In *Proceedings of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications*, pages 243–262, New York, NY, USA, 2009. Association for Computing Machinery.

4 Ana Milanova, Atanas Rountev, and Barbara G Ryder. Parameterized object sensitivity for points-to analysis for Java. *ACM Transactions on Software Engineering and Methodology*, 14(1):1–41, 2005.

5 Micha Sharir and Amir Pnueli. Two approaches to interprocedural data flow analysis. In S. S. Muchnick and N. D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 7, pages 189–234. Prentice-Hall, 1981.