Constrictor: Immutability as a Design Concept (Artifact)

Elad Kinsbruner[1]
Technion, Haifa, Israel

Shachar Itzhaky
Technion, Haifa, Israel

Hila Peleg
Technion, Haifa, Israel

Abstract

Many object-oriented applications in algorithm design rely on objects never changing during their lifetime. This is often tackled by marking object references as read-only, e.g., using the const keyword in C++. In other languages like Python or Java where such a concept does not exist, programmers rely on best practices that are entirely unenforced. While reliance on best practices is obviously too permissive, const-checking is too restrictive: it is possible for a method to mutate the internal state while still satisfying the property we expect from an "immutable" object in this setting. We would therefore like to enforce the immutability of an object's abstract state.

We check an object's immutability through a view of its abstract state: for instances of an immutable class, the view does not change when running any of the class's methods, even if some of the internal state does change. If all methods of a class are verified as non-mutating, we can deem the entire class view-immutable. We present an SMT-based algorithm to check view-immutability, and implement it in our linter/verifier, Constrictor.

We evaluate Constrictor on 51 examples of immutability-related design violations. Our evaluation shows that Constrictor is effective at catching a variety of prototypical design violations, and does so in seconds. We also explore Constrictor with two real-world case studies.

2012 ACM Subject Classification Software and its engineering → Software design engineering; Software and its engineering → Software defect analysis

Keywords and phrases Immutability, Design Enforcement, SMT, Liskov Substitution Principle, Object-oriented Programming

Digital Object Identifier 10.4230/DARTS.10.2.9

Related Article Elad Kinsbruner, Shachar Itzhaky, and Hila Peleg, "Constrictor: Immutability as a Design Concept", in 38th European Conference on Object-Oriented Programming (ECOOP 2024), LIPIcs, Vol. 313, pp. 22:1–22:29, 2024.
`https://doi.org/10.4230/LIPIcs.ECOOP.2024.22`

Related Conference 38th European Conference on Object-Oriented Programming (ECOOP 2024), September 16–20, 2024, Vienna, Austria

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2024 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

This artifact allows for the reproduction of tables 1, 2, 3, 4 from the paper.

[1] Corresponding author.

1.1 Table 1

Table 1 contains the results of running Constrictor on the Inheritance benchmark set. This benchmark set is divided into four hierarchies: lists, points, sets, and shapes. In order to reproduce these results, run `./create_table1.sh`. This should take less than 10 minutes. It is also possible to only run some of the hierarchies: `./create_table1.sh [-lists] [-points] [-sets] [-shapes]`. More than one argument can be provided. **The created table can be found at `results/table1.md`.** The script runs each benchmark 10 times (unless it times out) and the given table contains the average time for each benchmark.

Running the script on the Lists hierarchy takes 5 seconds on our computer. Points takes 30 seconds, Sets takes 348 seconds (5 minutes, 48 seconds) and Shapes takes 30 seconds.

1.2 Table 2

Table 2 contains the results of running Constrictor on the Non-inheritance benchmark set and the Aspects & Limitations benchmark set. In order to reproduce these results, run `./create_table2.sh`. This should take less than 20 minutes. **The created table can be found at `results/table2.md`.** The script runs each benchmark 10 times (unless it times out) and the given table contains the average time for each benchmark.

Note: Due to a strange bug in Z3, this script may crash in rare cases due to an assertion failure here, which may appear as a segmentation fault. In such cases, rerunning the script should work, and in any case this issue has not been observed when running the script on a specific benchmark as described below.

Clarification: The precision and recall given in the paper refer only to the Non-inheritance set, while the ones generated by the artifact refer to both sets.

Running the script takes 1136 seconds (18 minutes, 56 seconds) on our computer.

Both of the above scripts can also be used to directly run Constrictor on a specific benchmark from the relevant set by using the `-prog` option: `./create_table[1|2].sh -prog <BENCHMARK_NAME>`. In this case, the result is printed to stdout including specific details regarding which method caused a violation flag.

1.3 Table 3

Table 3 contains the results of the experiments described in Section 6.3, relating to the first RQ2 case study. In order to reproduce these results, run `./create_table3.sh`. This should take less than 5 minutes. **The created table can be found at `results/table3.md`.** It is also possible to recreate just one half of Table 3: **./create_table3.sh –part** [**a**|**b**].

Running the script takes 103 seconds (1 minute, 43 seconds) on our computer. Reproducing just the first part of Table 3 takes 89 seconds (1 minute, 29 seconds) and reproducing just the second part of Table 3 takes 25 seconds.

1.4 Table 4

Table 4 contains the results of the experiments described in Section 6.4, relating to the second RQ2 case study. In order to reproduce these results, run `./create_table4.sh`. This should take less than one minute. **The created table can be found at `results/table4.md`.**

Running the script takes 2 seconds on our computer.

2 Content

This artifact is an Ubuntu x86-64 Docker container containing the following folders at `/root`:

- `benchmarks`: Contains the benchmarks described in the paper. The benchmark sets are divided into four folders:
  - `inheritance_benchmarks` contains the benchmarks from the Inheritance set described in the paper, split by hierarchy.
  - `non_inheritance_benchmarks` contains the benchmarks from the Non-inheritance set described in the paper.
  - `kotlin_stdliblike_case_study` contains the classes comprising the Kotlin collections hierarchy discussed in Section 6.3 and shown in Figure 6(a), along with the relevant annotations as described.
  - `kotlin_stdliblike_case_study_fixed` contains the fixed hierarchy after applying the fix described in Section 6.3 and changing the hierarchy to that shown in Figure 6(b).
  - `red_green_trees_case_study` contains the three versions of the tree with bidirectional references described in Section 6.4.
- `constrictor`: Contains the implementation of the tool. This folder contains config.py which can be modified accordingly to change the number of program steps used in path unrolling (see Section 5). The location of the CVC5 executable used can also be configured here.
- `py2smt`: Contains the Py2Smt toolchain. It is an implementation component discussed in Section 5 and does not need to be changed for the operation of this artifact.
- `runners`: Contains the implementations of the scripts used to generate the tables for this artifact.
- `create_table{i}.sh`, for `i` $\in \{1, 2, 3, 4\}$: Scripts that can be used to easily run Constrictor on the provided benchmark sets in order to reproduce the three tables from the paper. The operation of these scripts is described below.
- `cvc5`: The binary for the CVC5 solver used for Constrictor (see Section 5). This is part of the Docker container but is not part of the artifact itself.
- `CVC5_LICENSE.txt`: The license by which CVC5 is made available.

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.10568857`.

4 Tested platforms

We ran the artifact for the figures in the paper on a 2022 MacBook Pro with an M2 processor and 16 GB of RAM. There is no significant disk usage and no need for a GPU. No other hardware resources are needed. The artifact is an Ubuntu x86-64 Docker container. We expect it to be runnable on any computer that can run such containers. The total run time for the reproduction of all four tables should be less than 90 minutes.

5 License

The artifact is available under Creative Commons License CC-BY 4.0.

6 MD5 sum of the artifact

1ce4f00b8fdd44404b88eb293f18f05d

7 Size of the artifact

0.87 GiB